1-1-1994

Use of virtual reality in off-line robot programming

Darren Scott Knapp
Iowa State University

Use of virtual reality in off-line robot programming

by

Darren Scott Knapp

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Mechanical Engineering
Major: Mechanical Engineering

[redacted] University
Ames, Iowa

1994

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my major professor Marty Vanderploeg for giving me the opportunity to work with him and making my graduate education enjoyable. I would also like to thank Judy Vance and Jeffrey Huston for serving on my committee.

In addition, I would like to thank Jim Troy and my other friends in the Visualization Lab for their assistance in my research.

Finally, I would like to thank my parents for their continued support throughout my many years in college.

1. INTRODUCTION

Off-line programming is the creation of a set of instructions for a robot to perform a task without using the actual robot. Many commercial software programs have been developed to program robots in this manner. Most software of this type utilize computer graphics to make the program more interactive. The advantages of using off-line programming and computer graphics include:

- Reducing downtime by not taking a robot out of service while it is being reprogrammed,
- Testing the robot program through computer graphic simulation,
- Fast developing and modifying of the robot program, and
- Increasing safety by testing the robot program through simulation rather than using the actual robot.

One definition of Virtual Reality is the use of computer graphics to generate a realistic interactive environment. The use of Virtual Reality devices such as the head mounted display (HMD) can provide the user the feeling of total immersion within the computer generated environment. Input devices such as a VPL DataGlove or spaceball allow the user to interact with the virtual environment.

The use of Virtual Reality in off-line programming of robots provides many additional features over conventional computer graphic based programming tools. Using a virtual environment provides a much more realistic representation of the robot workcell layout and robot simulation. This allows the operator to better understand the robot task and to

test the robot path by visually checking for events such as collision between the robot and its environment. Input devices such as the spaceball and DataGlove allows the operator to quickly define the robot path and also modify the workcell layout.

This thesis describes the development of off-line robot programming software which uses a virtual environment. This software, called *VRS* which stands for Virtual Robot Simulator, allows the user to program and simulate the Mitsubishi RV-M1 robot in an environment displayed using a HMD, stereo eyewear, or a standard computer monitor. *VRS* can utilize both a mouse and a spaceball as input devices. Once a task for the robot is programmed and simulated in *VRS*, a device control file may be written to operate an actual RV-M1 robot.

Chapter 2 of this thesis gives a literature review which presents the evolution of off-line robot programming including text-based computer programs, computer graphic-based programs, and the use of virtual environments. Chapter 3 outlines the theoretical development of *VRS* and presents the formulation for robot positioning using forward and inverse kinematics, robot path generation, and task simulation. Chapter 4 presents an overview of the functions and capabilities of *VRS*. Chapter 5 shows how *VRS* may be used to perform specific robot applications. Chapter 6 gives the results of tests which were based on the use of *VRS* to perform the applications in Chapter 5. Conclusions are presented in Chapter 7.

2. LITERATURE REVIEW

This literature review provides an overview of the evolution of off-line robot programming including text-based programming languages, graphic-based programming and simulation software, and the use of Virtual Reality in off-line robot programming and simulation.

2.1 Early Robot Programming Languages

When robots were first developed, the only way to program the robot to perform a certain task was to use a teach pendent. The operator would enter specific robot gripper locations and joint rotations directly into the memory of the robot system. The robot would then perform the task. Not only is this process slow and difficult, it creates downtime for the robot as it is being reprogrammed.

To decrease the amount of downtime for the robot, robot programming languages such as VAL II were developed. But languages of this type are not standard and are limited in their use. Later, AML was developed as a general purpose robot language[1]. These robot programming languages can be used with the teach pendent and give more efficient on-line programming. However, resulting output from the robot program can only be seen using the actual robot, limiting the robot in performing other tasks as the new program is being perfected.

2.2 Graphical-Based Robot Programming and Simulation

More recently, off-line graphic based programs have been developed to both program a robot to do a specific task and to see a simulation of the robot perform the task on a computer graphics terminal. Robot programming and simulation programs such as GRASP[2], CimStation[3], and IGRIP[4] offer advanced features to off-line robot programming such as collision detection and dynamic simulation. These programs offer the user many options concerning complexity, operating platform, and the type of robots being simulated.

Research has focused on making future robot simulation programs easier to use, more accurate, and useful for more types of robots. Smith[1] has proposed a new robot program, SMALL (Sawyer-motor Multi-robot Assembly workceLL), which has a graphical interface to change the various states of robot system and uses high-level C programming task functions such as 'pickupPart' which automatically tells the robot to go over to the selected object in the workspace and pick it up. Chen, Trivedi, and Bidlack[5] have developed an environment for simulation and visualization of sensor-driven robots. Neilsen, Trostmann, Trostmann, and Conrad[6] are implementing a new off-line robot programming and simulation system called ROPSIM. This system uses a neutral interface in order to provide a Computer Integrated Manufacturing and Engineering (CIME) system.

2.3 Virtual Reality in Robot Programming

The use of Virtual Reality (VR) or so-called virtual environments in robot programming is in its infancy. At this time no commercial software for programming robots in a virtual environment is available. Research in the area of off-line robot programming using Virtual Reality is not abundant, but is increasing.

Takahashi and Sakai [7] have proposed a way to program robot using Virtual Reality. Their method involves having the operator who is wearing a VPL DataGlove perform the required task in a virtual workspace which simulates the actual robot workspace. The system recognizes movement from the DataGlove and translates the movement into manipu-

lator commands for the robot. The robot then is able to perform the same task in the actual workspace. Takahashi and Ogata [8] modified this method of robot programming by making the hand movements recognizable as task-level commands. These task-level commands are then interpreted and converted into the manipulator-level commands to be used by the robot.

Research has also been conducted on using Virtual Reality in teleoperation of robots. Teleoperation is defined as operating a robot from a large distance. This distance may be defined as the distance to a robot which is in another laboratory or to a robot which is orbit about the earth. Operating a robot in this way may result in time delays between the input command and the robot reaction due to the communication linkage between the command center and the remote robot. These time delays make real time operation of the robot very difficult. To address some of the problems with teleoperation as well as off-line robot programming, Brunner, Heindl, Hirzinger, Landzettel[9] have developed a tele-sensor-programming concept that uses sensory perception to locally control the robot. This method uses sensor information such as distance, force-torque and vision sensors to give a correct representation of the real robotic behavior. This information is then used to simulate the robot in a virtual environment. Virtual environments have also been used in the development of robotic teleoperation for NASA's Space Station Freedom[10].

3. ROBOT KINEMATICS AND PATH GENERATION

3.1 Forward and Inverse Kinematics

Positioning the robot to perform a task requires finding a set of joint variables which will result in the having the manipulator at the desired location, orientation, and time. Either forward or inverse kinematics may be used to position the robot. Forward kinematics involves changing the joint variables to position the robot. The position and orientation of the manipulator can then be found based on the joint variables. The use of forward kinematics to achieve the desired manipulator position is difficult. It is not intuitive to see what joint values are needed to achieve the desired manipulator position. The use of inverse kinematics is a much more intuitive method to position the robot's manipulator. The manipulator's goal position is specified and inverse kinematics are used to derive the required joint angles. The mathematics required for the inverse kinematics solution is much more intense than forward kinematics. Forward kinematics involves link transformation matrix multiplication, while inverse kinematics requires solving non-linear systems of equations.

Virtual Robot Simulator, or *VRS*, uses both forward and inverse kinematics to change the position of the robot. Forward kinematics is utilized when a link of the robot is selected and rotated about its joint to a desired angle. Link rotation about a local coordinate point corresponding to the joint location and the use of link hierarchy simplify the forward kinematics. When a link is selected to rotate to a specified angle, that link and all other links which are in sequence to it down to the manipulator are oriented based on a 3x3 rotation matrix. This rotation matrix is based on the initial orientation of the robot

used in *VRS*. Figure 3.1 defines the joints and links of the RV-M1 robot [11].The initial orientation and frame location for the 5-DOF robot used is shown.in Figure 3.2.

When rotating about the waist, the body, upper arm, forearm, wrist and hand are given a new orientation based on a rotation about the Z1 axis shown in Figure 3.2. This can also be given as the rotation from frame 0 to frame 1, where frame zero is a fixed reference frame. This 3x3 rotation matrix is given as

$$ {}^{0}_{1}R = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

where the R signifies a rotation matrix. The subscript 1 and superscript 0 designate that the rotation is frame 1 relative to frame 0.

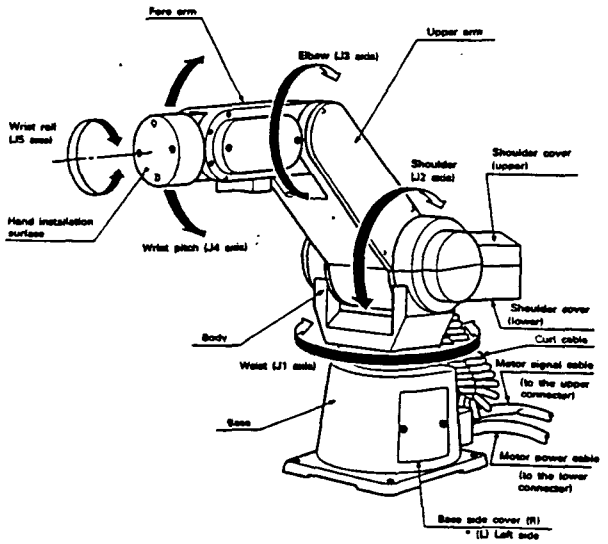

Figure 3.1 Link and joint definition of the RV-M1 robot.

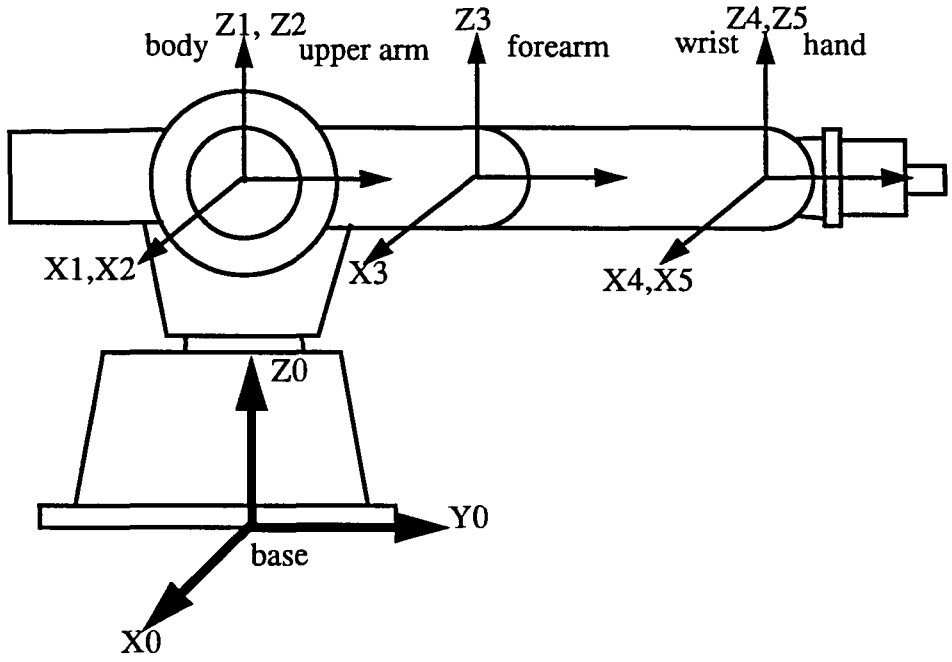

Figure 3.2 Initial joint orientation and frame location of the RV-M1 robot.

If the upper arm is selected as the link to be rotated, *VRS* automatically joins the forearm, wrist, and hand to the upper arm as lower levels of the hierarchy. These links are rotated about the X2 axis, which is the shoulder joint. The body link will remain unchanged. The rotation matrix for the local rotation of the upper arm about the shoulder joint is

$$ {}^{1}_{2}R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & -\sin\theta_2 \\ 0 & \sin\theta_2 & \cos\theta_2 \end{bmatrix} \tag{3.2} $$

The complete rotation matrix for the upper arm is a product of the waist rotation matrix and the upper arm rotation matrix and can be calculated as

$$ {}^{0}_{2}R = {}^{0}_{1}R\,{}^{1}_{2}R \qquad (3.3) $$

The following equations give the rotation matrices for the forearm rotating about the elbow, and the hand rotating about the wrist pitch joint and the wrist roll joint, respectively.

$$ {}^{2}_{3}R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_3 & -\sin\theta_3 \\ 0 & \sin\theta_3 & \cos\theta_3 \end{bmatrix} \qquad (3.4) $$

$$ {}^{3}_{4}R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_4 & -\sin\theta_4 \\ 0 & \sin\theta_4 & \cos\theta_4 \end{bmatrix} \qquad (3.5) $$

$$ {}^{4}_{5}R = \begin{bmatrix} \cos\theta_5 & 0 & \sin\theta_5 \\ 0 & 1 & 0 \\ -\sin\theta_5 & 0 & \cos\theta_5 \end{bmatrix} \qquad (3.6) $$

As shown before, all rotation is relative to frame 0 and link rotation matrices must be premultiplied to get the rotation matrix relative to frame 0.

The inverse kinematics formulation used in VRS is based on the method used by Troy[12] for a Mitsubishi RV-M1 robot. This method was developed using Denavit-Hartenberg [13] parameters as a basis for defining the coordinate transformation matrices. These coordinate transformation matrices relate one link of the robot to another and also are used to define the location of the tool relative to the base. Figure 3.3 gives the frame assignments for the RV-M1 robot based on the D-H parameters.

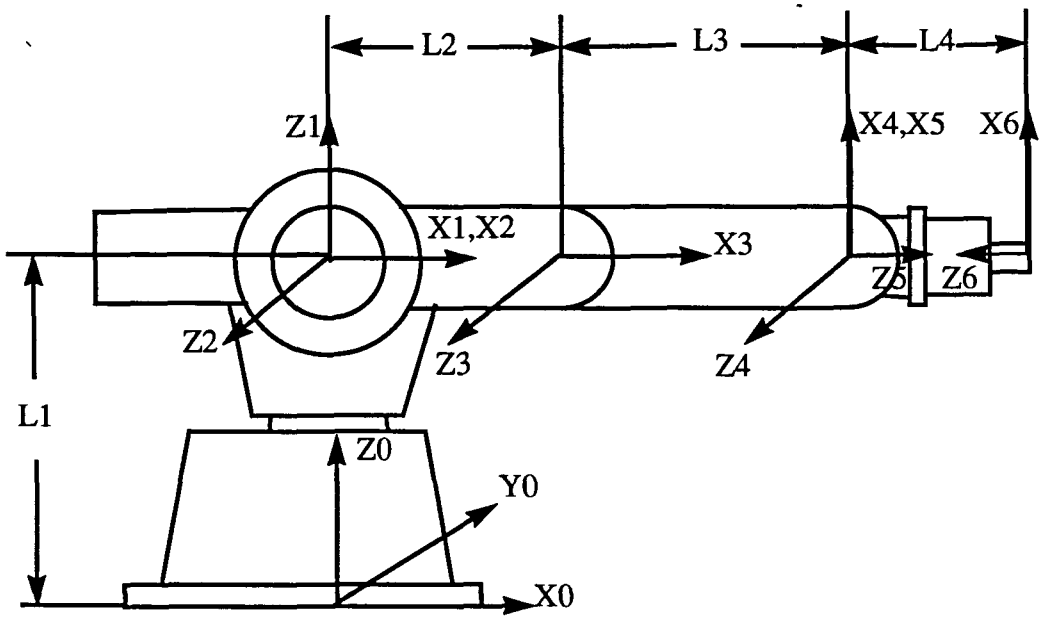

Figure 3.3 Frame assignments of the RV-M1 robot.

Equations 3.7 to 3.12 give the frame to frame coordinate transformation matrices. These transformation matrices are used to determine the transformation of the wrist frame relative to the base frame. The wrist relative to base transformation is given by Equation 3.13.

$$ {}^{0}_{1}T = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7} $$

$$ {}^{1}_{2}T = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8} $$

$$ {}^{2}_{3}T = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9} $$

$$ {}^{3}_{4}T = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & L_3 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10} $$

$$ {}^{4}_{5}T = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11} $$

$$ {}^{5}_{6}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12} $$

$$ {}^{B}_{W}T = {}^{0}_{5}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T \tag{3.13} $$

The position of the robot's tool relative to a stationary point is required. Position of the tool relative to the wrist and the base relative to a stationary point is specified by the user. The base frame and the stationary frame may or may not be coincident. The transformation matrix of the tool relative to the station is given by Equation 3.14.

$$ {}^{S}_{T}T = {}^{S}_{B}T\,{}^{B}_{W}T\,{}^{W}_{T}T \tag{3.14} $$

The ${}^{S}_{T}T$ transformation matrix is used to determine the tool location based on joint variables from forward kinematics.

To perform inverse kinematics for the 5-DOF robot, the desired X, Y, and Z tool location is specified for the ${}^{S}_{T}T$ transformation matrix. The ${}^{B}_{W}T$ transformation matrix is extracted from ${}^{S}_{T}T$ using equation 3.15.

$$ {}^{B}_{W}T = {}^{S}_{B}T^{-1}\,{}^{S}_{T}T\,{}^{W}_{T}T^{-1} \tag{3.15} $$

Once the ${}^{B}_{W}T$ transformation matrix is calculated the joint angles can be found using a closed form solution approach. The derivation of the joint angles from this transformation matrix is presented in Troy[12].

3.2 Path Generation

Path generation is the defining of the trajectory that the robot's manipulator will follow when performing a task. The path is generated based on control points which define the manipulator's position and orientation at a point during the execution of the task. The complete path is calculated by joint angle interpolation from one control point to the next. For this thesis the joint angle interpolation is based on the use of polynomial splines defined by supplied robot orientation points. The algorithm used in *VRS* was developed by Troy[12]. A path that the robot's manipulator is to follow can be broken up into

smaller path segments. Each segment is characterized by a start and end point. At the start and end points the manipulator's velocity is zero. The splined cubic polynomials used in the joint interpolation satisfy this requirement. Splined cubic polynomial are used to connect the control points along the path. The polynomials are solved based on the conditions that start and end points have zero velocity and a specified maximum acceleration. A complete derivation of the solution to the polynomials used in joint interpolation is given in Troy[12].

3.3 Task Level Programming

Task level programming is the automatic development of a robot path to perform simple tasks such as pick and place of workcell objects. Utilizing task level programming can greatly decrease the time needed for programming robots to do a very complex task. This can be accomplished by breaking down the complex task into smaller tasks which can be automatically developed. Task level operations which are available in *VRS* are placing a workcell object at a new location and orientation, placing an object on top of another object based on the orientation of the second object, and grasping a specified object.

The first step in a pick and place operation for the 5-DOF RV-M1 robot is rotating the robot about its waist to face the object which is to be picked up. The resulting waist angle in radians for the robot when its stationary frame coincides with the base frame is given as

$$\theta_1 = \text{atan}\frac{y}{x} - \frac{\pi}{2} \tag{3.16}$$

where x and y are the location of the workcell object in the stationary frame. Figure 3.4 shows the definition of θ_1. Forward kinematics is then used to rotate the gripper down to a vertical position. This is accomplished by setting the shoulder, elbow, and wrist roll angle to 0 degrees and the wrist pitch angle to -90 degrees. This orientation ensures that the manipulator is in a vertical position which will ultimately allow it to grip the workcell

object. Inverse kinematics is then used to translate the gripper over to the objects location. During this translation the orientation of the gripper is unchanged. Workcell objects are brought into *VRS* with their local coordinate frame in the same orientation as the stationary frame of the robot. Any rotation of an object about its local Z axis can be found from its rotation matrix. The rotation matrix of an object which is confined to rotate by an angle Ψ about its local Z axis is given in equation 3.17.

$$R = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

From this matrix the rotation angle is calculated. Care must be used when calculating the angle in order that it ends in the correct quadrant, yielding the correct sign for the angle. The wrist roll angle necessary to align the gripper with the object is given in Equation 3.18.

$$\theta_5 = \psi + (\theta_1 + \frac{\pi}{2}) \tag{3.18}$$

The gripper is then lowered to an appropriate height based on the maximum extent of the object in the Z direction. The gripper is closed and the object is moved up an appropriate distance. The robot is rotated about its waist to an angle based on the X and Y location of the object's destination. The object and gripper are then translated to the exact location of the object's destination. The gripper is then rotated to a new angle based on the objects desired final orientation. The object is lowered to its final position and released from the gripper. The gripper is raised from the object and closed, completing the pick and place task.

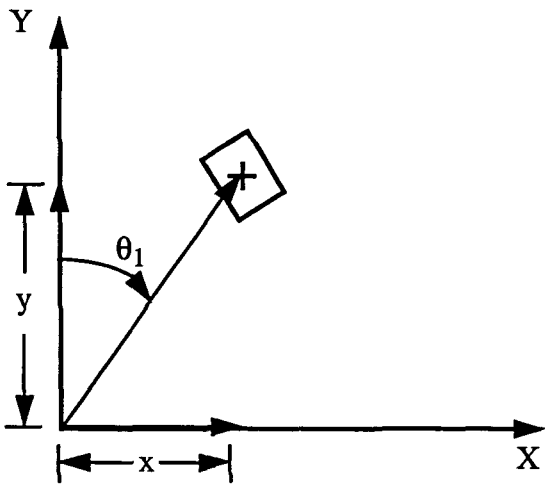

Figure 3.4 Definition of θ_1.

Positioning an object on top of another object uses the same algorithm as before, except that the object's final location and orientation is based on the orientation of a second object. The minimum extent of the first object and the maximum extent of the second object along the Z axis is compared to ensure that collision between the two objects is avoided when the first object is placed on the second.

4. VIRTUAL ROBOT SIMULATOR

Virtual Robot Simulator, *VRS*, is a computer program developed for this thesis to study the advantages of using Virtual Reality (VR) in off-line robot programming and simulation. Currently, off-line robot programming software is difficult to use. *VRS* is not only very easy to use, but also has display and interface capabilities not found in current robot programming and simulation software. The *VRS* program utilizes Virtual Reality technology. Head mounted displays (HMD), VR booms, and stereo viewing eyewear are supported by *VRS* to provide a virtual robot workcell environment in which to program the robot. Input devices such as the spaceball can be used to simplify the programming process. These special capabilities of *VRS* provide an working environment in which the user is able to quickly program a robot to perform a task and observe the task being performed by the robot in a realistic virtual environment. Once a robot program has been successfully developed and simulated, a device control file for the robot can be automatically created to be used with the actual robot. Figure 4.1 shows a schematic of the available equipment used in off-line robot programming with *VRS*.

4.1 General Requirements for VRS

4.1.1 Software Requirements

VRS is a program developed using WorldToolKit software from Sense8, Inc.[14]. WorldToolKit is a library of C subroutines which are used to develop computer programs capable of using a variety of user interface devices to create a virtual environment.

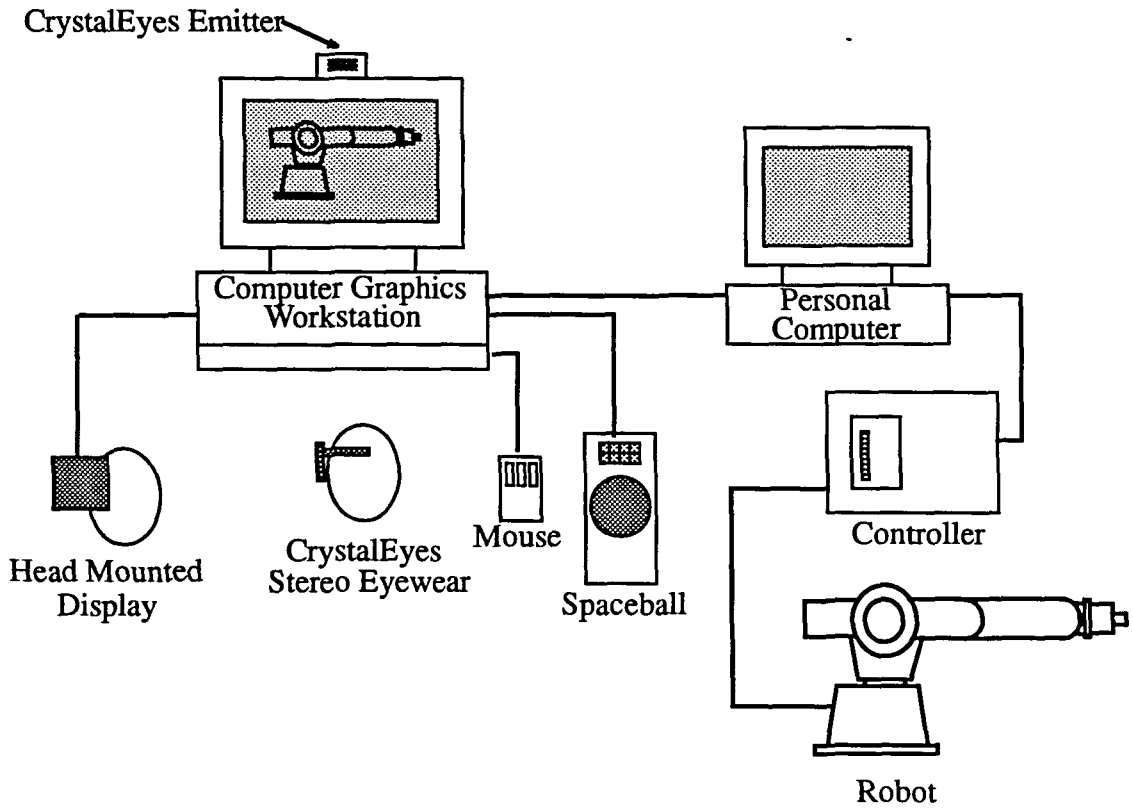

Figure 4.1 Schematic for VRS off-line programming.

Interface devices which are supported by WorldToolKit include the following:

1. Standard mouse
2. Spaceball Technologies Spaceball.
3. CiS Geometry Ball Jr.
4. Fake Space Labs BOOM, BOOM2C, and BOOM3C.
5. Logitech 3D Mouse and Head Tracker.
6. Ascension BIRD and FLOCK of birds.
7. Polhemus ISOTRAK and FASTRAK sensors.
8. StereoGraphics CrystalEyesVR with head tracker.
9. Analog and GRAVIS MouseStick optical joystick (PC only).

WorldToolKit is also capable of using GL subroutines from Silicon Graphics, Inc.[15] Because *VRS* uses both WorldToolKit and GL subroutine calls, these software packages must be accessible to use *VRS*. Currently, WorldToolKit is available for about $8000.

4.1.2 Solid Modeling Requirements

To create a robot and its workspace to be simulated using *VRS*, the models must be in a format familiar with WorldToolKit. WorldToolKit is able to use two different solid model format types: dxf and nff (neutral file format). The nff files represent a solid model as a listing of numbered vertices in 3D space and a listing of connectivity which connects the vertices into simple polygons. These polygons can then be assigned a material or color which determines how the object will appear on the computer screen. Textures, which are colored images, can also be assigned to the polygons. Textures are useful in representing complex objects in computer graphics. An example is taking photographs of different sides of a building and using them as textures. These textures may be assigned to the polygons of a simple cube, transforming the cube to a realistic representation of the building.

The nff file format is used in *VRS*. This format has a listing of vertices, connectivity of each polygon, color of the polygon, textures used on the polygons, and orientation of those textures.

The robot and the workspace was first modeled using the I-DEAS Solid Modeling software from SDRC. From I-DEAS the solid models were written out in the universal file format. A converter was then used to change the universal file into an nff file. The converter accepted polygon color, shading type, and texture information to be used in the model.

4.2 VRS Capabilities

This section presents the capabilities of *VRS*. All user input into the program is done by picking various objects in the environment or selecting various buttons to perform specific functions. If the spaceball is used as the input device, the function buttons of the spaceball are used to select the specific function. The user interface was restricted to selecting function buttons and picking objects in order to try to preserve the sense of being in a virtual world and also eliminating the need for the user to need any type of keyboard input. While in a virtual world which requires the use of a head mounted display, the user would have to remove the HMD in order to be able to input commands from the keyboard. Having to remove the HMD would remove the sense of being in the virtual world.

The functions of *VRS* are divided into different subsystems which are accessible from the main menu. Figure 4.2 shows the main menu of *VRS*. The following sections outline the functions and capabilities of each subsystem.

When *VRS* is used with a 2D mouse, the mouse is used to both move the current viewpoint and pick objects. The middle mouse button is used to switch between these two functions. An indicator at each subsystem informs the user which mode the mouse is currently using. When in the pick objects mode, the mouse can be used to move the cursor to an object and the left mouse button is pressed to pick the object or button. When the mouse is in the move viewpoint mode, the cursor position and the mouse buttons dictate how the viewpoint is changed. Table 4.1 summarizes the use of the mouse to change the viewpoint.

4.2.1 Display Subsystem

The display system is used to change display effects such as the amount of ambient light and convergence distance and parallax values for stereo viewing. Sensor sensitivity can also be changed in this subsystem. Figure 4.3 shows the display subsystem.

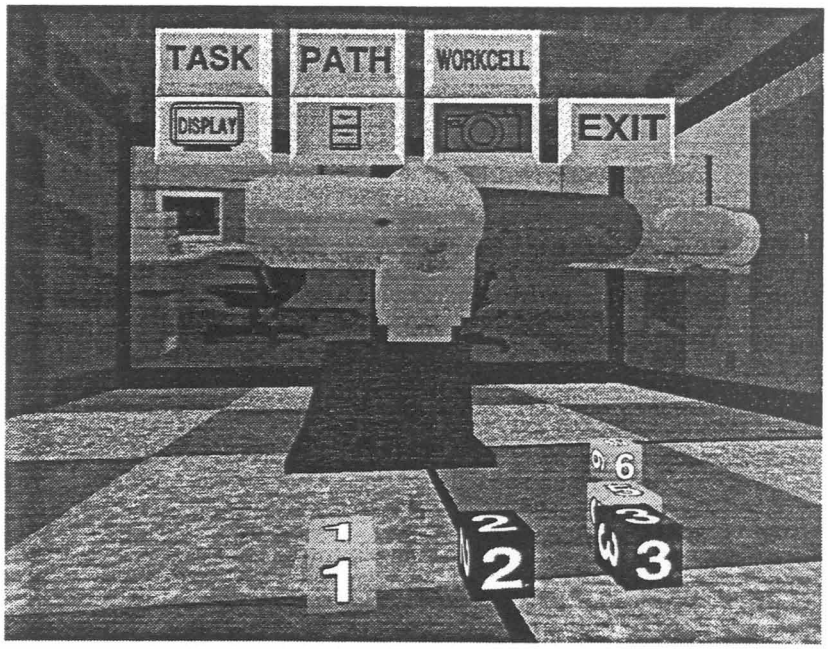

Figure 4.2 Main menu of VRS.

Ambient light is light that illuminates the surfaces of workcell objects regardless of their position or orientation. The amount of ambient light used can be increased or decreased by 10% by selecting the ambient light button and then the up or down arrow button. Directional lights are automatically incorporated into the program by a file entitled "lights.lts". This file designates the X, Y, Z, coordinates of the light position, the X, Y, Z components of the light direction vector, and the light intensity. A sample lights.lts file is given in the Appendix.

Stereo viewing of *VRS* can be accomplished by setting the computer monitor to stereo display and using CrystalEyes stereo eyewear. This is done automatically if the CrystalEyes option is invoked when executing *VRS*. Two values which effect how well the stereo effect is presented is the parallax value and convergence distance. The parallax value is the distance in the virtual world which the left and right eye views are drawn. Convergence distance is the distance at which the stereo image is perceived to exist. Varying this value may cause the image to appear behind or in front of the monitor, giving very inter-

Table 4.1 Changing the viewpoint with the mouse.

MOUSE BUTTON(S)	SCREEN LOCATION	VIEWPOINT EFFECT
LEFT	TOP	MOVE FORWARD
LEFT	BOTTOM	MOVE BACKWARDS
LEFT	LEFT	YAW LEFT
LEFT	RIGHT	YAW RIGHT
RIGHT	TOP	MOVE UP
RIGHT	BOTTOM	MOVE DOWN
RIGHT	LEFT	PAN LEFT
RIGHT	RIGHT	PAN RIGHT
LEFT AND RIGHT	TOP	PITCH UP
LEFT AND RIGHT	BOTTOM	PITCH DOWN
LEFT AND RIGHT	LEFT	ROLL LEFT
LEFT AND RIGHT	RIGHT	ROLL RIGHT

esting effects. The parallax value is increased or decreased by 10% when the parallax button and the up or down button are selected. The convergence distance can be changed by +/- 10 mm when selected.

Sensor sensitivity is the maximum magnitude a sensor will translate the viewpoint or an object which is being translated with the sensor. The mouse and spaceball are sensors which can have their sensitivity adjusted. Selecting the mouse or spaceball sensitivity button and the up or down arrow will cause the sensitivity for the sensor to increase or decrease by 10%. Related to sensitivity is angular rate. The angular rate of a sensor is the maximum angular rate of change about an axis. This value can also be changed by 10% by selecting either the mouse or spaceball angular rate button and then the up or down button.

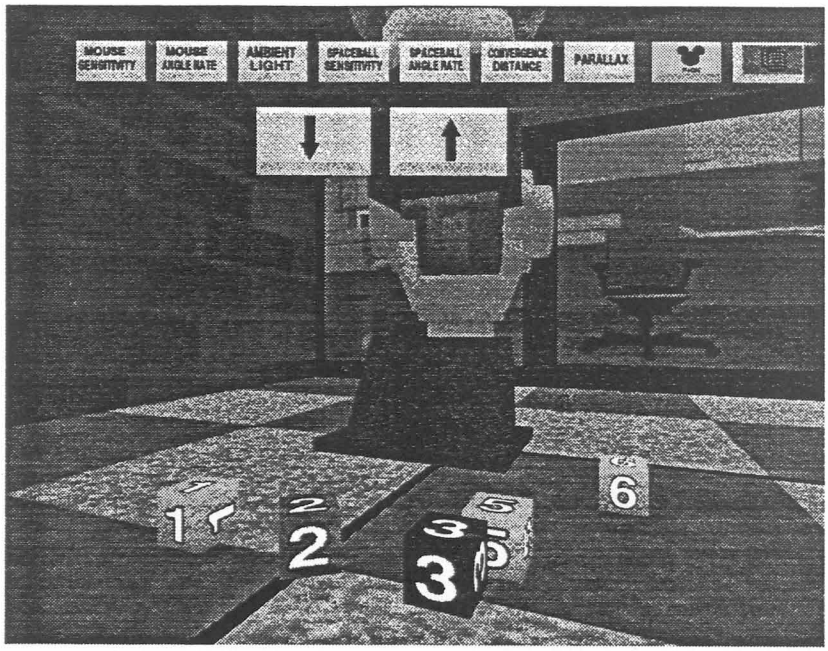

Figure 4.3 Display subsystem

4.2.2 Path Subsystem

The path subsystem is used to create a path for the robot to follow. This subsystem controls the robots forward kinematics, inverse kinematics, path definition, and task simulation. The path definition is aided by the use of intersection detection between two objects of the workcell and between a workcell object and the floor. The robot's gripper is moved to the desired location and orientation using forward and inverse kinematics of the robot. When the robot is at a desired location the complete orientation of the robot at that point is recorded. After various key locations and orientations along the path are recorded, the completed path is computed. A simulation of the robot performing the task can then be observed. Figure 4.4 shows the layout of the path subsystem.

4.2.2.1 Forward Kinematic Control. The orientation of the robot can be changed by selecting a link of the robot and rotating it plus or minus the needed number of degrees.

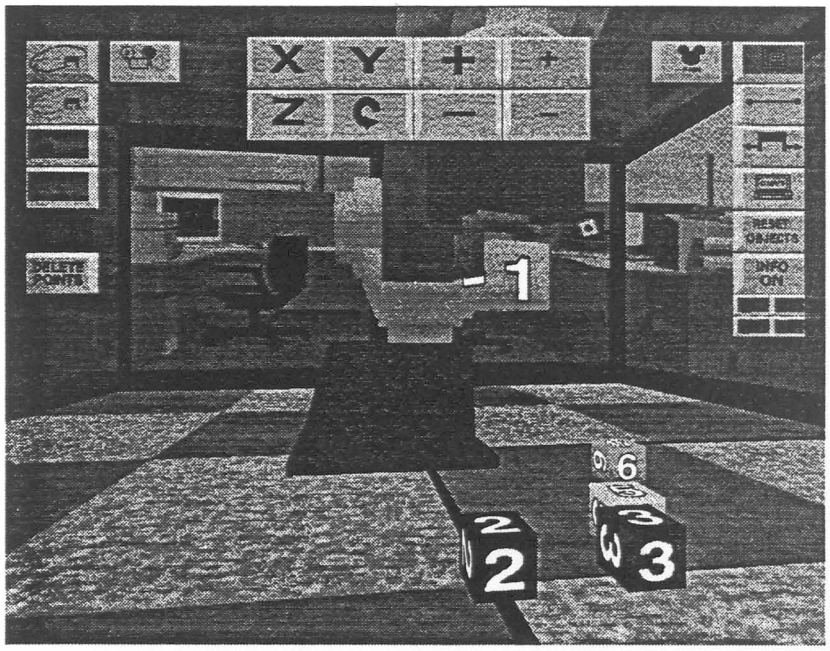

Figure 4.4 Path subsystem.

The links that can be manipulated are the body, the upper arm, the forearm, the wrist, and the hand. When one of these links is picked, the color of the link turns red. The rotation button, indicated by the curved arrow is then selected. Selecting this button causes the button to turn green indicating it is ready to perform a joint rotation. Changing the joint angle is accomplished by selecting one of four buttons: plus, fine plus, minus, and fine minus. These buttons are indicated by the large and smaller plus and minus buttons. When selecting the larger plus button, the link as well as the lower members of the robot will rotate about the picked link's joint in the positive direction. For example, if the body and the positive button is selected, the body, upper arm, forearm, wrist, and hand will rotate about the body's joint, which is the waist, in the positive direction. If the forearm is selected, the forearm, wrist, and hand will move in the positive direction about the elbow. The rotation of the robot's link will continue until one of the four rotation direction buttons is selected. The plus and minus rotation buttons give a joint rotation in plus and

minus one degree increments, respectively. The fine plus and minus buttons give a joint rotation in plus and minus 1/100 degree increments.

4.2.2.2 Inverse Kinematic Control. Because gripper position is a resultant of the joint angles of the links, simple forward kinematics is used to determine where the gripper is positioned and oriented based on the known input of joint rotation angles. However, it is very difficult to achieve a desired gripper position by joint rotation alone. Direct gripper translation is used to solve this problem. Given a desired gripper position, inverse kinematics is used to determine the required joint angle to result in the desire gripper position.

To perform the gripper translations one of the three translation buttons is selected. These buttons are indicated by X, Y, and Z. By selecting one of these buttons and one of the four positive/negative buttons, the gripper is translated in the positive/negative X, Y, or Z direction. The translation continues until one of the four plus/minus buttons is selected. Translation is resumed when one of these buttons is once again selected. The plus and minus buttons translate the gripper in +/-1.0 mm increments and the small plus and minus and negative buttons translate the gripper in +/-0.1 mm increments.

Two basic robot orientations are the home and origin positions. The home position is the position the actual robot is set to when it is idle. The origin position is the position of the robot when all joint angles are set to zero. In *VRS* the robot may automatically be set to one of these positions by selecting either the home position or origin position button. These buttons are designated by images of the robot at these two positions.

To see the current status of the robot the information on/off button can be selected. When the information on button is selected, a data line consisting of the X, Y, and Z location of the gripper and the joint angles of the robot are displayed in the lower portion of the screen or display device. If the gripper is currently gripping a workcell object, the X, Y, and Z location and the objects rotation about the Z axis is displayed.

Once the gripper is in a location to pick up an object, it can be opened using the gripper open/close button indicated by an image of the robot gripper. Once the gripper is open

the gripper can be translated into position to grip the object. The gripper close button is then selected. The gripper will the close around the object using intersection detection between the object and both fingers of the gripper. The contact surfaces of the gripper are colored red to simplify orienting the gripper to a good position to grasp the object. When the gripper is closed around a object, that object is then automatically attached to the gripper and hand. The object will then move with the gripper and hand until the gripper is opened, freeing the object.

4.2.2.3 Path Definition. In order to simulate a robot performing a task, key robot positions along the path of the robot are recorded as control points. A cubic spline is then computed connecting the selected control points to create a complete path. The control points of the path are entered by orienting the robot into the desired position and then selecting the start/end button. The start/end button is used at the beginning and end of desired segments within the path as well as before and after the opening or closing of the gripper. Points are selected until the complete path is defined.

Selecting the compute button will compute the complete path used by the simulation. Robot orientations are calculated for each frame of the simulation. The length of time the robot takes to complete the path is based on the specified maximum speed and acceleration. The number of frames or stored robot orientations computed is set at 30 frames per second.

Once the path is computed, the complete path may be seen by selecting the path display button indicated by the gripper and solid path button. The robot path and the gripper position is displayed. The original control points can be seen by selecting the control point display button indicated by a control points and gripper representation.

The delete points button of the path subsystem deletes all the control points of the current path. The current simulation may still be viewed until the compute button is selected, resetting the simulation. The menu button indicated by the menu icon in the upper right

corner may be selected to exit the path subsystem. By choosing this button the main menu will reappear.

4.2.2.4 Path Simulation. The simulation of the robot performing the task can be seen by selecting the movie button. By selecting this button all of the path subsystem buttons disappear and a control panel much like a VCR appears. The greater than/less than buttons control play and reverse, respectively. Likewise, the much greater than, much less than buttons control fast forward and rewind. The simulation can be advanced or stepped backwards one frame at a time with the plus and minus frame buttons. The center button stops the simulation at any point. Selecting the movie button again removes the control panel and retrieves the path subsystem. Figure 4.5 shows the movie control panel.

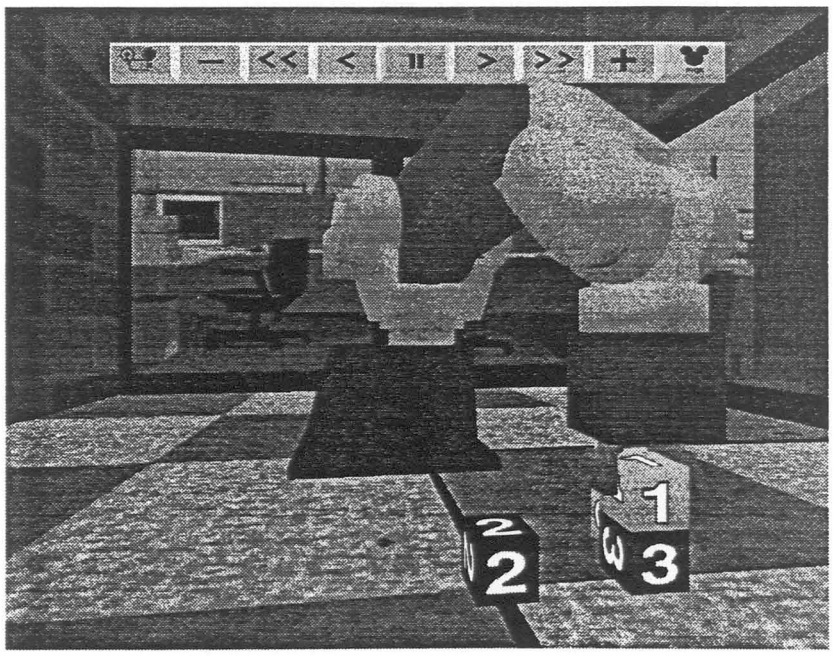

Figure 4.5 Movie control panel in the path subsystem.

4.2.2.5 Intersection Detection. The generation of an acceptable path for the robot to perform a task is aided by the use of intersection detection. The intersection detection determines when collisions between two workcell objects or between a workcell object and the floor occurs. If when positioning a workcell object that is in the robot's gripper collides with the another object or the floor, the motion of the gripper and the object is stopped. The gripper's motion is then reversed until the collision is no longer detected. This collision detection helps when positioning workcell objects on top of one another and placing the objects on the floor.

4.2.3 Task Subsystem

The task subsystem is used to greatly simplify the creation of a path for the robot to do simple tasks. These simple tasks include pick and place of a workcell object and placing of one object on the top of another object. In the path subsystem, in order to pick and place a workcell object, the user is required to do the following:

1. Move the robot's gripper over to workcell object,
2. Rotate the gripper into a position which will grip the object,
3. Open the gripper,
4. Move the gripper down to the object, stopping at an appropriate height relative to the object,
5. Close the gripper,
6. Move the gripper and object up and over to the placing position,
7. Move the gripper and object down to the surface, stopping at the appropriate height,

8. Open the gripper,

9. Move the gripper up and out of the way of the object, and

10. Move the robot to a final position.

All of these steps require inserting the appropriate start/end points into the path. Fine tuning of the gripper position is often necessary to pick up the object. This requires centering the gripper precisely over the object and aligning the gripper with the edges of the object. The task subsystem eliminates all of these steps. Figure 4.6 shows the task subsystem.

To do a simple pick and place task in the task subsystem the workcell object is picked. A bounding box is then created on the object. The PUT button is then selected. This signifies that the current picked object is the active object. The TO button is selected next, signifying that a pick and place operation will be performed. Once this button is selected, the X, Y, Z, rotation button panel appears and a copy of the active object is made. This new object may be translated and rotated into the final placing position using the button panel. Selecting the GO button completes the task assignment. After the GO button is selected the active object will move to its final position.

Placing an object on top of another is very similar to the pick and place routine described before. The object to be moved is picked and a bounding box is shown around that object. The PUT button is selected, making the current picked object the active object. The object to which the active object is to be placed upon is picked. The ON button is then selected signifying the current object as the passive object. The GO button is then selected completing the task.

If it is desired to have the robot go grasp an object but not move it, the grasp button may be selected after the desired object is picked. The grasp function is a very useful tool when used in conjunction with the path subsystem to program very complex tasks.

Once the task is defined, the path subsystem is used to simulate the task. The object reset button resets the workcell objects to their original positions. The compute button computes the path. The simulation of the robot can be seen using the VCR control buttons. Any number of pick and place tasks may be appended to one another. Paths generated in the path subsystem can be appended to tasks created in the task subsystem creating a more complex robot task.

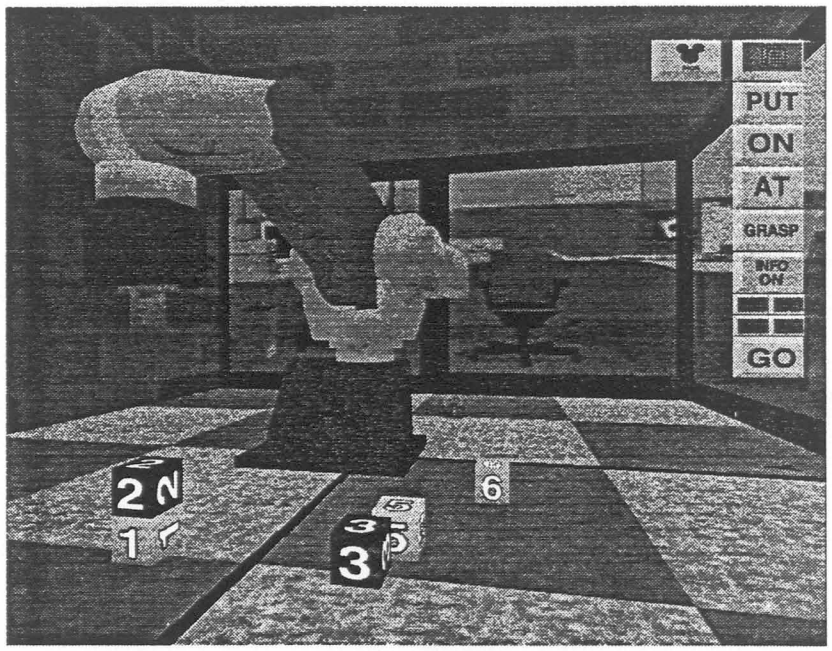

Figure 4.6 Task subsystem

4.2.4 Workcell Subsystem

The workcell subsystem is used for setting the material or color of the robot, workcell objects and background. It is also used to position the objects in the workcell. Figure 4.7 shows the workcell subsystem.

Once a workcell object or robot link is picked to be modified, a bounding box appears on the object which was selected. The color of the picked object is modified by changing

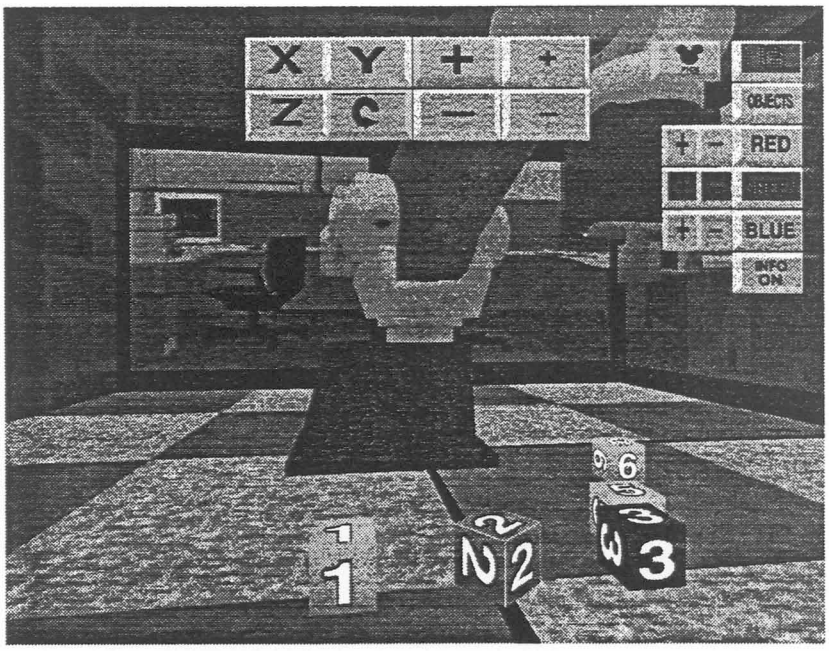

Figure 4.7 Workcell subsystem

the red, green, and blue values of the color. When one of these buttons is picked, plus and minus indicators appear. Picking these indicators increase and decrease the corresponding color component by one. WorldToolKit currently allows sixteen values for each component ranging from zero to fifteen, giving a total of 4096 color combinations. A zero value of red, green, and blue produces black and a value of fifteen for each component produces white. If the plus button is increased beyond the value of fifteen, that value is reset to zero. To change the color of the background the workcell/background button is selected. The background may then be changed in the same manner as the workcell objects.

The position and rotation about the Z axis can be modified for the workcell objects. The translation and rotation panel of buttons used for the path subsystem are also used to perform the positioning and rotation of the workcell objects. After the object to be positioned is chosen, the X, Y, Z, or rotation button and a plus or minus button is picked. This will cause the object to translate or rotate until a plus or minus button is selected again.

The position and rotation angle of the selected object can be display by picking the information button.

4.2.5 Camera Subsystem

The camera subsystem is used to create and modify camera locations or viewpoints. This subsystem is reached by selecting the camera button in the main menu. Figure 4.8 shows the camera subsystem.

Currently there are four available cameras, three of which can be modified. The fourth camera is designated as the gripper camera. This camera moves with the gripper giving an interesting perspective to the viewer as the robot is performing a task. The mouse is used to change the viewpoint of the environment. The set camera button for the camera being modified is then picked. This saves the current viewpoint as the camera's new viewpoint. The four cameras can also be selected within the path and task subsystem.

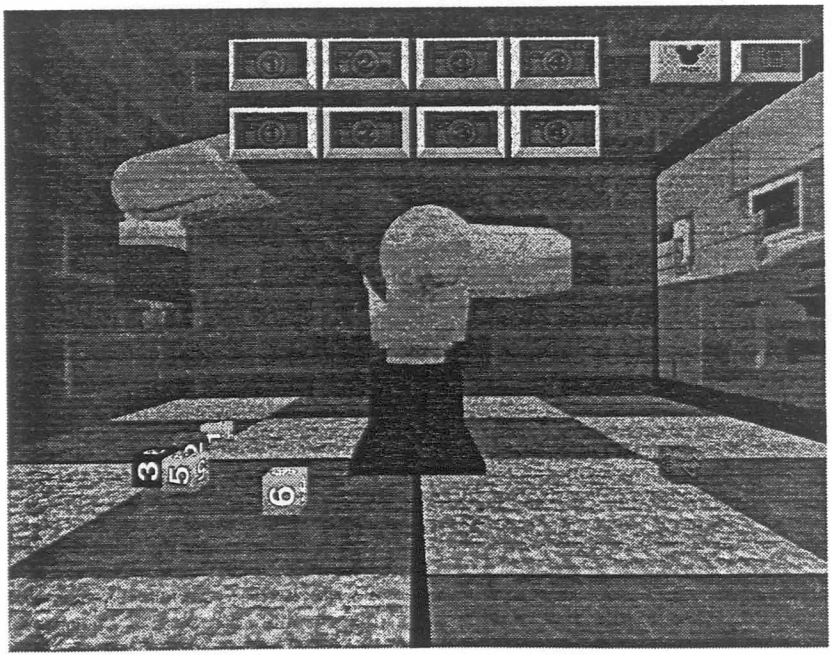

Figure 4.8 Camera subsystem.

4.2.6 File Input/Output Subsystem

The file input/output subsystem is used to read in or write out points files of a computed robot task, status files of the workcell environment, and a device control file for the actual RV-M1 robot. The points file is a data file defining the orientation and gripper information of the robot for each key point of the task. After a robot task has been defined and computed, the write points button can be selected to write out the points file. The file written will be given the generic name of 'points.pts' in order to minimize keyboard use. The read points file button reads in a file called 'points.pts'. After this file is read in the robot path is computed in the path subsystem and the task may then be simulated. The 'points.pts' file may be changed to a more appropriate name and later changed back to 'points.pts' in order to read the file into *VRS*. The workcell write button writes out a file called 'workcell.obj' containing the stored initial positions, orientations, and materials of the workcell objects. Workcell objects may be given stored positions, orientations, and materials by selecting the workcell read button when the appropriate file is copied to 'workcell.obj'. The RV-M1 write button writes out a file named 'rvm1.bas' which can be used with the actual RV-M1 robot. Figure 4.9 shows the file input/output subsystem.

4.2.7 Spaceball Operation

The spaceball may be used with *VRS* instead of the mouse to execute the desired commands. A spaceball is a computer input device consisting of a palm sized ball which detects 6 different directional inputs from the hand. These 6 inputs or degrees of freedom are X, Y, and Z translational input and roll, pitch, and yaw rotational input. The spaceball also has 8 function buttons on the device and a pick button on the front face of the ball. *VRS* uses the spaceball and the function buttons to select different subsystems, change viewpoints, move the robot's gripper, and orient the workcell objects. The spaceball can be used to orient and position the gripper of the robot in defining a path in the path subsystem. When the spaceball is used for robot orientation, translational input to the spaceball will cause the gripper to move within its environment in the same direction as the in-

Figure 4.9 File input/output subsystem.

put. For example, left or right movement of the spaceball with cause the gripper to move toward the left or right direction of the viewpoint. Pitch and roll of the spaceball will change the orientation of the gripper by changing the wrist pitch and roll angles. The spaceball can also be used to change the position and orientation of the workcell objects in the workcell and task subsystems. Spaceball roll input will cause the selected workcell object to rotate about the Z axis. Table 4.2 summarizes the use of the spaceball in *VRS* for gripper and workcell object orientation. Table 4.3 defines the *VRS* functions of the spaceball buttons. Table 4.4 shows how the spaceball is used for changing the viewpoint.

Table 4.2 Spaceball use for gripper and workcell object orientation

SUBMENU	SPACEBALL INPUT	EFFECT
PATH	LEFT, RIGHT	GRIPPER TRANSLATION IN X, Y DIRECTIONS
	UP, DOWN	GRIPPER TRANSLATION IN Z DIRECTION
	ROLL	WRIST ROLL
	PITCH	WRIST PITCH
WORKCELL	LEFT, RIGHT	OBJECT TRANSLATION IN X, Y DIRECTIONS
	UP, DOWN	GRIPPER TRANSLATION IN Z DIRECTION
	ROLL	OBJECT ROTATION ABOUT Z AXIS
TASK	LEFT, RIGHT	OBJECT TRANSLATION IN X, Y DIRECTIONS
	UP, DOWN	GRIPPER TRANSLATION IN Z DIRECTION
	ROLL	OBJECT ROTATION ABOUT Z AXIS

Table 4.3 Spaceball button operation of *VRS*.

SUBSYSTEM	BUTTON	EFFECT
MAIN MENU	1	SELECTS PATH SUBSYSTEM
	2	SELECTS TASK SUBSYSTEM
	3	SELECTS WORKCELL SUBSYSTEM
	4	SELECTS CAMERA SUBSYSTEM
	5	SELECTS DISPLAY SUBSYSTEM
	6	SELECTS FILE I/O SUBSYSTEM
	8	SELECTS EXIT
PATH	1	PICKS START/END POINT
	2	GRIPPER OPEN/CLOSE
	3	COMPUTE PATH
	4	RESET OBJECTS' POSITIONS
	5	INFORMATION ON/OFF
	6	MOVIE SUBSYSTEM
	7	DELETE ALL POINTS
	8	MAIN MENU
	PICK	VIEWPOINT/ROBOT ORIENTATION
MOVIE	1	BACK ONE FRAME
	2	REVERSE PLAY
	3	FORWARD PLAY
	4	FORWARD ONE FRAME
	5	STOP
	6	PATH SUBSYSTEM
	8	MAIN MENU

Table 4.3 (continued.)

SUBSYSTEM	BUTTON	EFFECT
TASK	1	PUT ACTION
	2	AT ACTION
	3	ON ACTION
	4	GRASP ACTION
	5	GO ACTION
	8	MAIN MENU
	PICK	VIEWPOINT/ROBOT ORIENTATION
WORKCELL	1	RED COMPONENT UP
	2	GREEN COMPONENT UP
	3	BLUE COMPONENT UP
	4	OBJECT/BACKGROUND COLOR
	5	RED COMPONENT DOWN
	6	GREEN COMPONENT DOWN
	7	BLUE COMPONENT DOWN
	8	MAIN MENU
	PICK	VIEWPOINT/WORKCELL ORIENTATION
CAMERA	1	CAMERA 1
	2	CAMERA 2
	3	CAMERA 3
	4	CAMERA 4
	5	SET CAMERA 1
	6	SET CAMERA 2
	7	SET CAMERA 3
	8	MAIN MENU
	PICK	VIEWPOINT/WORKCELL ORIENTATION

Table 4.3 (continued.)

SUBSYSTEM	BUTTON	EFFECT
DISPLAY	1	SPACEBALL SENSITIVITY
	2	SPACEBALL ANGULAR RATE
	3	CONVERGENCE DISTANCE
	4	PARALLAX
	5	AMBIENT LIGHT
	6	DECREASE VALUE
	7	INCREASE VALUE
	8	MAIN MENU
	PICK	VIEWPOINT/WORKCELL ORIENTATION
FILE I/O	1	WRITE POINTS FILE
	2	WRITE WORKCELL FILE
	3	WRITE RV-M1 DEVICE CONTROL FILE
	5	READ POINTS FILE
	6	READ WORKCELL FILE
	PICK	VIEWPOINT/ROBOT ORIENTATION
EXIT	1	CONFIRM EXIT
	2-8	ABORT EXIT

Table 4.4 Spaceball use for changing viewpoint

SPACEBALL INPUT	VIEWPOINT EFFECT
LEFT	VIEWPOINTS MOVES LEFT
RIGHT	VIEWPOINT MOVES RIGHT
UP	VIEWPOINT MOVES UP
DOWN	VIEWPOINT MOVES DOWN
ROLL	ROLL VIEWPOINT
PITCH	PITCH VIEWPOINT
YAW	YAW VIEWPOINT

5. VRS APPLICATION EXAMPLES

Examples of using *VRS* to program and simulate a RV-M1 robot are given in this chapter. The examples give step by step instruction on how to use *VRS* to complete the given robotic task. In these examples, six blocks are placed in the workcell. The blocks are to be moved to a initial position. The robot is then programmed to construct a pyramid with the blocks. In the first example the path subsystem is used to program the robot. The task subsystem is used in the second example. The mouse is used as the input device and CrystalEyes stereo eyewear are used as the display device.

5.1 Initial Setup

Before the *VRS* program is executed, some directories and files need to be present. The directories contain the geometric models of the robot, workcell objects, *VRS* function buttons, and textures. A dynamic lighting file is also necessary. In the /rvm1 directory are nine files which make up the model of the RV-M1 robot. Separates files for the various links of the robot are required because each link must be able to move independently from the other links. The files which make up the robot are the following:

1. base.nff
2. shoulder.nff
3. upperarm.nff
4. forearm.nff
5. wrist.nff

6. hand.nff
7. finger1.nff
8. finger2.nff
9. robotfloor.nff

The workcell objects which are to be used *VRS* must be contained within the same directory. When executing *VRS*, this directory is specified by typing -d and the name of the directory.

The /textures directory contains the rgb texture files used in *VRS*. These textures are primarily for the buttons in *VRS*, but any textures used by the workcell objects are also found here. In order for WorldToolKit to find this directory, the environment variable WTIMAGES must be set. This is done by typing the following at the prompt:

> setenv WTIMAGES (path)/textures

The /buttons directory contains the buttons used in *VRS*. No changes need to be made to this directory.

The file 'lights.lts' contains the lighting files used in *VRS*. This file may be edited to preference. An example lights.lts file in given in the Appendix with an explanation of the file format.

5.2 Pyramid of Blocks

This section gives a step by step instruction on how to use *VRS* to create a pyramid of blocks. Six blocks will be loaded into the workcell. The blocks will be moved to specified locations. First the robot is programmed by to build the pyramid using the path subsystem. The robot is then reprogrammed using the task subsystem. Output files will then be created to save the block positions and the key robot orientations used for generating

the path of the robot. A device control file will also be created so that the task may be performed using an actual RV-M1 robot.

The options available when executing *VRS* are:

-c Crystaleyes stereo eyewear.

-s Spaceball.

-i Intersection detection.

To execute *VRS* and use the Crystaleyes eyewear and enable intersection detection type the following:

> vrs -denvironment/blocks -c -i

When the program begins the robot and six blocks are present. The MAIN MENU is also present. Figure 5.1 shows the main menu and the initial position of the blocks.

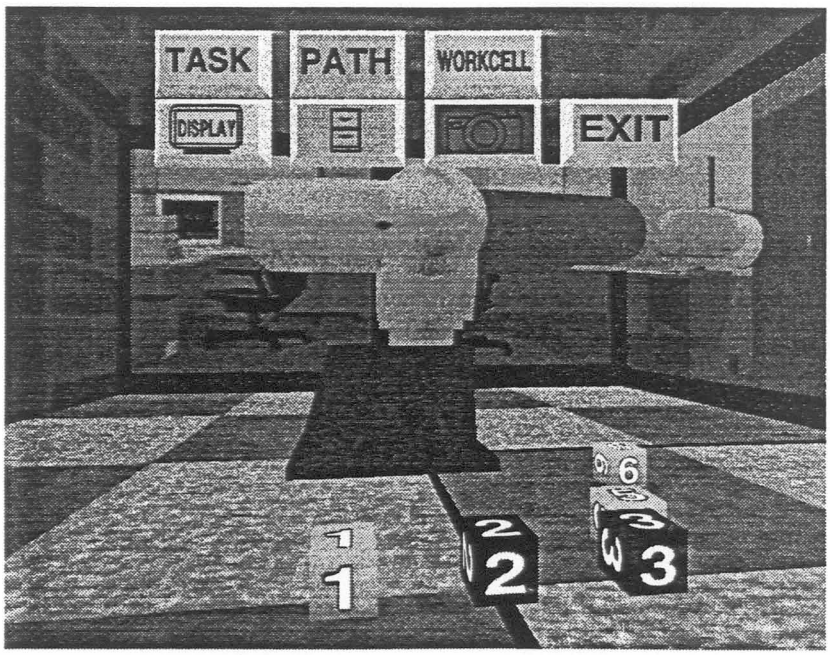

Figure 5.1 Main menu and initial position of the blocks.

5.2.1 Display Adjustment

The DISPLAY subsystem is selected first. Convergence distance and parallax values can be adjusted to give the best image through the stereo eyewear. This is done by selecting either the CONVERGENCE DISTANCE or PARALLAX buttons and the increase or decrease buttons. The sensitivity and angular rate of the mouse may also be adjusted in the same manner.

5.2.2 Camera Modification

The viewpoint may be changed to give a better view of a location within the workcell. This is done by pressing the middle mouse button to put the mouse in the viewpoint change mode. The mouse can now be used to change the viewpoint. The middle mouse button must be pressed again to use the function buttons or to pick workcell objects. A red or green indicator in the upper right corner of the screen shows the current operation mode of the mouse.

Cameras move the current viewpoint of the workcell to a predefined viewpoint. Go to the CAMERA subsystem to change the initial camera positions. There are four available cameras. The initial camera assignments have Camera 1 looking down the negative X axis, Camera 2 looking down the negative Y axis, and Camera 3 looking down the positive X axis. The fourth camera is designated as the gripper camera and is fixed to the gripper of the robot. The mouse may be used to change the viewpoint to a desired location. Select the SET CAMERA 1 button to assign the viewpoint to this camera. Cameras 2 and 3 may also be changed.

5.2.3 Workcell Object Color Assignment and Positioning

The main menu button in the upper right hand corner is selected to go back to the main menu. The WORKCELL subsystem is selected next. In this subsystem the blocks can be repositioned and can change color.

- Select the workcell object BLOCK 1. A white bounding box appears signifying that it is the current object to be modified.

- Change the red, green, and blue value of the block by selecting the RED, GREEN, and BLUE buttons and selecting the respective PLUS and MINUS buttons. The color of any block and the robot linkages can be changed.

- Select the BACKGROUND button. The background color may now be changed using the color change buttons. Selecting the OBJECTS button allows the workcell objects to have their color changed again.

For this example the six blocks are to have an initial position and orientation as defined below:

1. BLOCK 1, X= 125, Y= 350, Z= 0, Z axis rotation of 45 degrees
2. BLOCK 2, X= 0, Y= 350, Z= 0, no Z axis rotation
3. BLOCK 3, X= -100, Y= 350, Z= 0, Z axis rotation of -60 degrees
4. BLOCK 4, X= 125, Y= 250, Z= 0, no Z axis rotation
5. BLOCK 5, X= 0, Y= 250, Z= 0, no Z axis rotation
6. BLOCK 6, X= -100, Y= 250, Z= 0, no Z axis rotation

- Pick BLOCK 1. Now select the INFORMATION ON button. This shows the current position of BLOCK 1.

- Select the X button to move BLOCK 1 to its initial position. Now select the MINUS button. BLOCK 1 will begin to move in the negative X direction.

- Select one of the PLUS or MINUS buttons again when the X position value of BLOCK 1 is near 100. This stops the block from moving.

- Select the FINE MINUS or FINE PLUS button to fine tune the position of BLOCK 1, stopping it when its X position value reaches 100.

- Change the Y position of the block the same way by first selecting the Y button.

The Z position value does not have to be changed. It should be noted that *VRS* will not allow workcell objects to fall below the floor or off of the work table.

The orientation of BLOCK 1 is to be 90 degrees. Select the ROTATION button and rotate BLOCK 1 until the Z axis rotation value is 90 degrees. Position and orient blocks 2 through 6 in the same way. Figure 5.2 shows the blocks at their new positions.

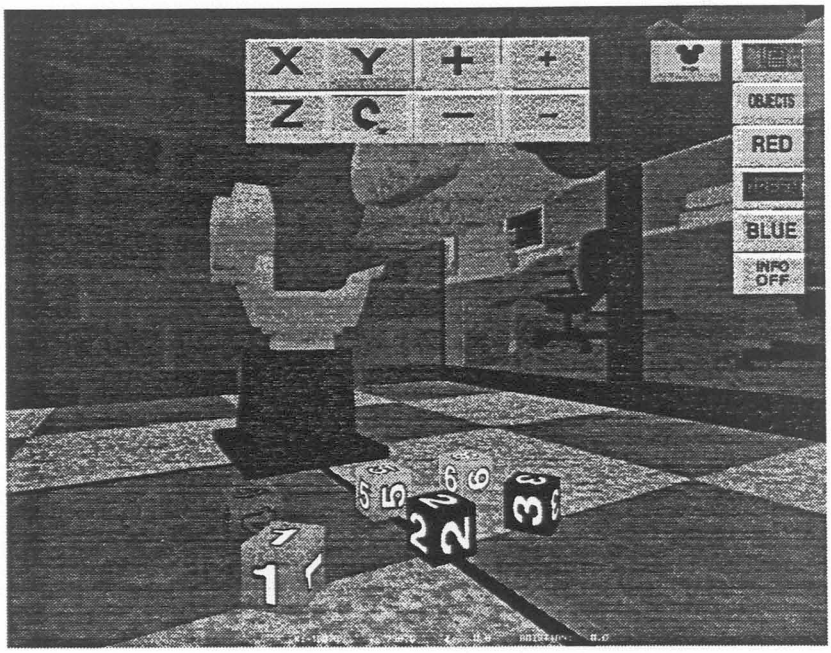

Figure 5.2 New positions of the blocks.

5.2.4 Block Pick and Place

To build the pyramid using the six blocks, the final positions of the blocks are to be as follows:

1. BLOCK 1, X= 300, Y= -60, Z= 0, no Z axis rotation
2. BLOCK 2, X= 300, Y= 0, Z= 0, no Z axis rotation

3. BLOCK 3, X= 300, Y= 60, Z= 0, no Z axis rotation
4. BLOCK 4, X= 300, Y= 30, Z= 40, no Z axis rotation
5. BLOCK 5, X= 300, Y= 30, Z= 40, no Z axis rotation
6. BLOCK 6, X= 300, Y= 0, Z= 80, no Z axis rotation

The next step is to develop a path for the robot to pick and place the blocks to create a pyramid. The initial position for the robot will be the home position.

- Select the HOME POSITION button. This moves the robot to the home position.

- Select the START/END button to make the home position the first point of the path.

- Select the POINTS DISPLAY button. This button turns on the enumeration of the start/end points. A number 1 will appear at the location of the first point. As each start/ end point is defined its point number will appear at that location.

- Rotate the robot about its base over to BLOCK 1. This is done by picking the body of the robot, selecting the ROTATION button, and then the PLUS button.

- Select the PLUS button again to stop the robot at the appropriate angle.

- Select the START/END button to save this position as the next key point of the path.

- Rotate the upper arm and then the wrist down until the wrist and hand of the robot is in a vertical position.

- Translate the gripper to the position directly over BLOCK 1 by using the X, Y, PLUS, and MINUS buttons. The INFORMATION ON button can be selected to see the current gripper position.

- Rotate the hand so that the gripper will line up with the two opposing faces of the block. Select the START/END button.

- Open the gripper by selecting the OPEN GRIPPER button. Select the START/END button.

- Move the gripper down to an appropriate height around BLOCK 1 by using the Z and MINUS buttons. Select the START/END button.

- Close the gripper and select the START/END button. The gripper will close until it detects intersection between the gripper fingers and BLOCK 1. Workcell object BLOCK 1 is now part of the gripper and hand and will move with the gripper and hand as a unit.

- Move the gripper and block up and away from the other blocks and select the START/END button.

- Rotate the robot and translate the gripper to a position over the final position of the BLOCK 1. Select the START/END button.

- Rotate the gripper until the block is at its correct final orientation and select the START/END button.

- Translate BLOCK 1 down to its final position and select the START/END button.

- Open the gripper and select the START/END button.

- Move the gripper up and out of the way of BLOCK 1 and select the START/END button. Close the gripper and select the START/END button.

This completes the picking and placing of the first block. Pick and place the remaining blocks in the same way, inserting the start/end points at the appropriate locations. The green camera buttons may be used to aid the positioning of the blocks by providing a better viewpoint. Figure 5.3 shows the use of the path subsystem in building the pyramid.

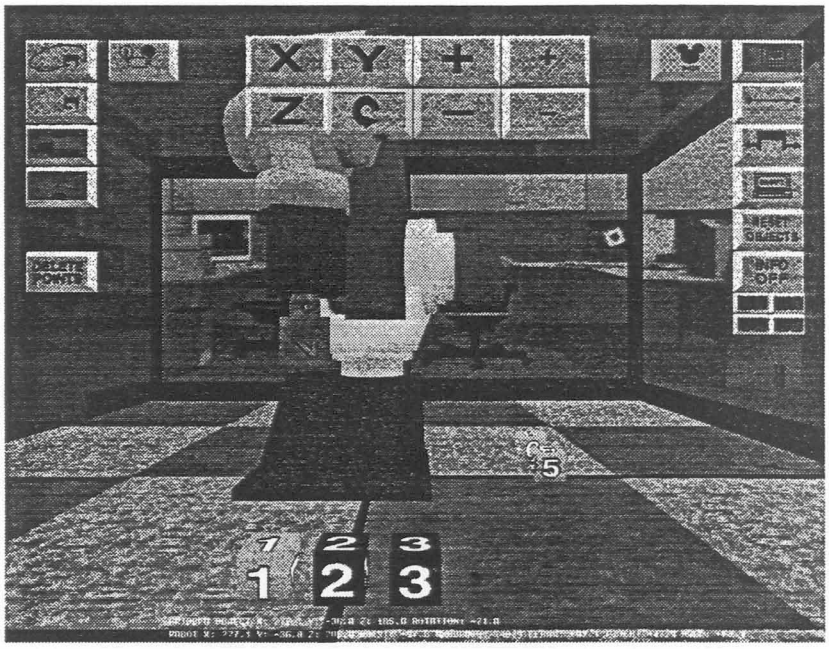

Figure 5.3 Building the pyramid using the path subsystem.

5.2.5 Simulation of the Robot

The next step is to compute the path of the robot.

- Select the COMPUTE button.
- Reset the blocks to their initial position using the RESET OBJECTS button.
- Select the PATH DISPLAY BUTTON. The path of robot gripper will be displayed. Moving the viewpoint gives an interesting 3 dimensional perspective of the robot's path.
- Select the MOVIE button.

The PATH subsystem buttons disappear and are replaced by the VCR buttons. The robot is moved to the orientation defined at the first start/end point. Selecting the forward play button sets the simulation in motion. A frame counter at the bottom of the display show the current frame number. At the end of the simulation the reverse button may be selected to run the simulation from the end to the beginning. The simulation may be sus-

pended at any time by selecting the STOP button. The FAST FORWARD and REWIND buttons play the simulation at twice the normal speed. Selecting the MOVIE button again returns the PATH subsystem. Figure 5.4 shows a frame from the robot simulation. The final pyramid in shown in Figure 5.5.

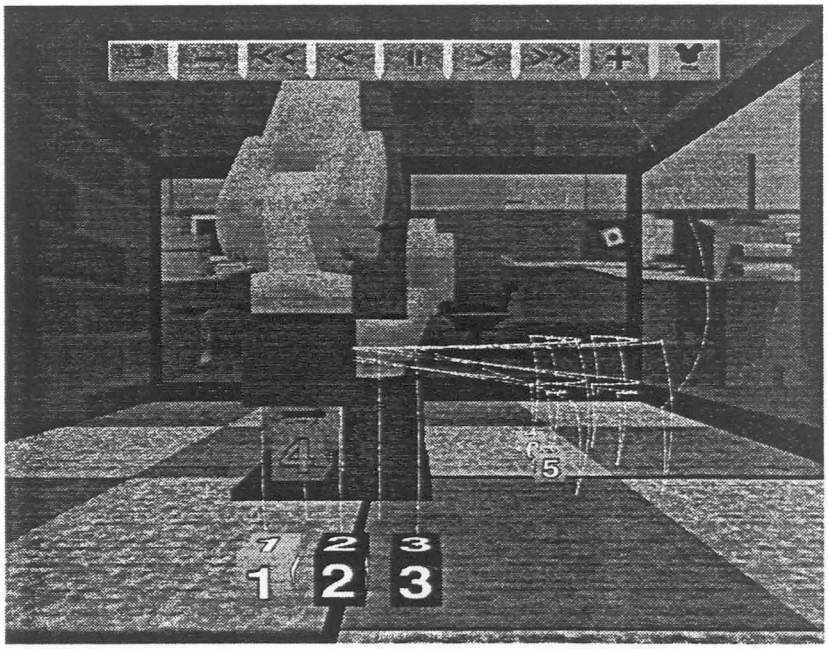

Figure 5.4 Simulation of the robot building the pyramid.

5.2.6 Saving the Simulation

- Select the FILE I/O button from the main menu to enter the file input/output sub-system.

- Select the POINTS WRITE button to write out a file called 'points.pts' which contains the orientation of the robot at each defined START/END point for the simulation.

- Select the RV-M1 WRITE button to write a file called 'rvm1.bas' used by the RV-M1 robot.

- Select the WORKCELL WRITE button to write out the initial positions and materials of the workcell objects into a file called 'workcell.obj'.

Examples of these files are given in the Appendix.

5.2.7 Running a Saved Simulation

To read in a points file copy the wanted file into 'points.pts' and the corresponding workcell objects to 'workcell.obj'. Select the POINTS READ and WORKCELL READ buttons.

Select the COMPUTE button from the path subsystem. The robot path will be computed. The RESET OBJECTS button resets the workcell objects to their new initial positions. The simulation may now be viewed using the VCR control buttons. More segments may be added to the path by defining more start/end points.

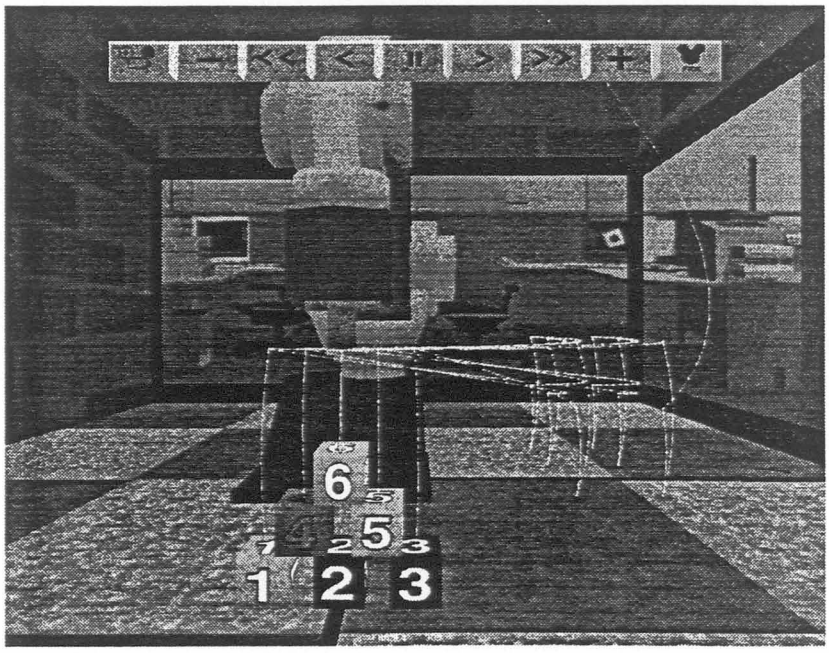

Figure 5.5 Final pyramid of blocks.

5.2.8 Using the Task Subsystem.

The task of building the pyramid using six blocks will be repeated using the task subsystem instead of the path subsystem. It will quickly become apparent that using the task subsystem is a much faster way to perform simple pick and place operations than using the path subsystem.

- Start *VRS* as before and orient the blocks to their new locations using the WORKCELL subsystem.

- Select the TASK subsystem.

- Pick BLOCK 1 and select the PUT button and then the AT button.

This states that BLOCK 1 will be picked and placed at a new location. A copy of BLOCK 1 has been created. This duplicate block will designate the final location of BLOCK 1. The same buttons used to position and orient a workcell object in the workcell subsystem appear when the AT button is selected. These buttons are used to move the duplicate block to the final position. Use these buttons to move the block to the final position and select the GO button. The duplicate block is then removed and replaced with BLOCK 1.

- Move the remaining five blocks in the same way as was done for BLOCK 1 to complete the pick and placing of the block for the pyramid. Figure 5.6 shows the use of the task subsystem as it is used to build the pyramid.

- Go to the PATH subsystem and select the HOME POSITION button and the START/END button to define the home position as the final position of the robot in the simulation.

The robot performing this task can now be simulated by first selecting the COMPUTE button and using the VCR control buttons.

Robot tasks may be defined by using the task subsystem in conjunction with the path subsystem. The task subsystem may be used to perform the simpler pick and place portions of the task and the path subsystem may be used to perform the more complex path definitions.

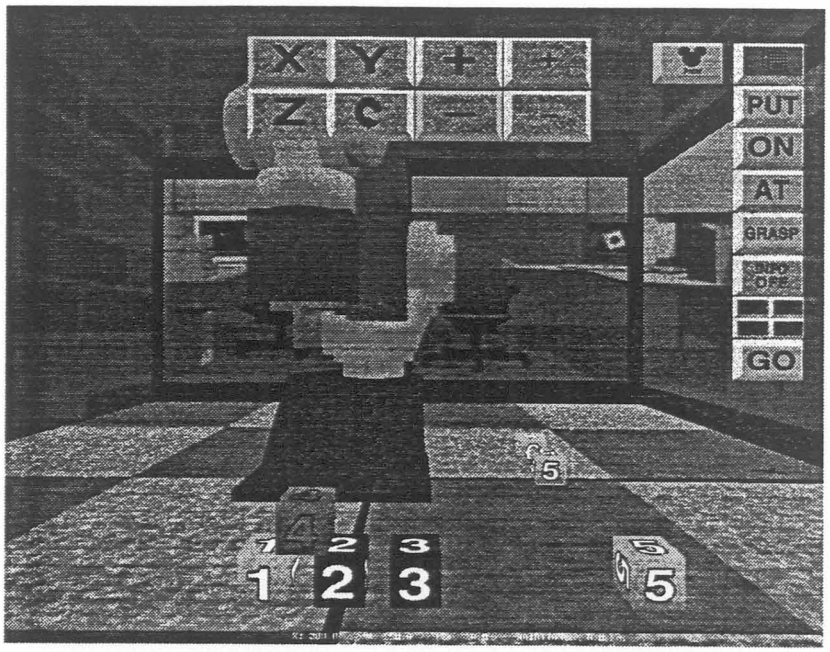

Figure 5.6 Using the task subsystem to build the pyramid.

6. RESULTS

To test the various capabilities of *VRS*, two students were selected to learn *VRS* and perform the application examples found in Chapter 5. This application required the user to build a pyramid of blocks using to different approaches; the path subsystem and then the task subsystem. The students were asked to perform the applications using different display and input devices. The students recorded how long it took to perform the application using specified display and input devices. The students were also asked to describe the strengths and weaknesses of using a certain device with *VRS*.

Table 6.1 shows the recorded times to perform the application in Chapter 5. Times for building the pyramid using the path subsystem and the task subsystem are shown. For this limited sample, the task subsystem is more efficient than using the path subsystem for this application. The reduced completion times for the task subsystem were present regardless

Table 6.1 Application completion times in minutes.

Display Device	Input Device	Path Subsystem	Task Subsystem
Monitor	Mouse	57.5	4.3
Stereo eyewear	Mouse	52.5	4.9
Monitor	Spaceball	60.0	7.0
Stereo eyewear	Spaceball	44.5	6.4

of which input and display devices were used. The use of the stereo eyewear decreased completion times for most of the cases. Using the spaceball was not as efficient as the mouse in completing the task, except when using the spaceball with the stereo eyewear.

Many advantages and disadvantages of using a certain display or input device were reported by the students. Some of the advantages for using the mouse are that it is used in most computer programs, therefore easy to use, and that it is quick and precise in picking objects. However, when using the CrystalEyes stereo eyewear, picking objects became more difficult. This is due to the way the monitor displays the graphics in the stereo mode. Other disadvantages are that it takes many mouse picks and movements to program the robot and that changing the viewpoint is often difficult.

A major advantage for using the spaceball is that moving the robot is more intuitive. Moving the robot's gripper with the spaceball eliminates the many button picks with the mouse which would be required to move the robot. However, it is difficult to isolate the movement of the robot's gripper to one direction when using the spaceball. For example, it is difficult to move the robot's gripper in the X direction without also moving it in the Y direction. Changing the viewpoint with spaceball was considered more difficult than with the mouse. We believe that once the spaceball function keys are memorized, the spaceball will be a faster and more powerful input device for *VRS* than the mouse.

Advantages for using the CrystalEyes eyewear is that the stereo viewing helps in the placing of the workcell objects and in gauging the position of an object relative to another object. As stated before, using the CrystalEyes eyewear makes picking of objects more difficult.

Some improvements of *VRS* that were suggested are simplifying the task and path subsystems by reducing the number of commands necessary and creating a way to quickly change to predefined viewpoints when using the spaceball. Another suggestion was displaying a projection of the gripper's location onto the floor to show the location of the gripper relative to an object which is to be gripped.

7. CONCLUSION

This thesis presented the development of off-line robot programming software which uses several different types of Virtual Reality technology. Virtual Robot Simulator (*VRS*) is capable of using Virtual Reality display devices such as a head mounted display and stereoglasses, and interface devices such as the spaceball. The use of a virtual environment created a realistic robot workcell environment in which the operator can quickly program a robot to do a specific task. The simulation of the robot performing the task was found to be greatly enhanced by using the virtual environment. In addition to the Virtual Reality capabilities of *VRS*, task level path generation and collision detection enabled faster and more precise off-line robot programming.

The use of *VRS* to program a robot to build a pyramid of blocks was tested. It was found that the use of the task subsystem greatly reduced the time needed to program the robot compared to using the path subsystem. The uses of Crystaleyes stereo eyewear helped in the positioning of the workcell objects and in relating the location of one object to another. However, the use of the stereo eyewear hindered the ability to pick objects and buttons using the mouse. The mouse was a more efficient input device than the spaceball. This can be attributed to the fact that the mouse is a more common input device and is easier to use when initially learning to use *VRS*. We believe that once a user becomes more comfortable in using the spaceball and the user memorizes which spaceball function key performs a command in *VRS*, the spaceball will be a faster and more powerful input device than the mouse.

Future development of *VRS* includes the following:

1. Expand *VRS* to include the capability to program more types of robots,
2. Incorporate the use of a data glove to program a robot in *VRS*,
3. Include dynamic analysis of the robot performing a task, and
4. Addition of a model of the RV-M1 teach pendent so that *VRS* can be used as a way to teach new operators how to use the RV-M1 robot.

BIBLIOGRAPHY

[1] Smith, M. "An Environment for More Easily Programming a Robot." Proceedings of the 1992 IEEE International Conference on Robotics and Automation, v 1, pp 10-16, May 1992.

[2] Derby, S. "GRASP From Computer Aided Robot Design to Off-line Programming." *Robotics Age*, v 5, n 2, pp 11-13, February 1984.

[3] SILMA Inc. "The CimStation User's Manual," Version 4.0, Available from SILMA Inc., 1601 Saratoga-Sunnyvale Rd., Cupertino, Calif., 95014, 1989.

[4] Mogal, J. S. "IGRIP - a Graphics Simulation Program for Workcell Layout and Off-line Programming." Robots 10 Conference proceedings, published by Robots International of the SME, pp 65-77, 1986.

[5] Chen, C. Trevedi M., Bidlack C. "Simulation and Graphical Interface for Programming and Visualization of Sensor-based Robot Operation." Proceedings of the 1992 IEEE International Conference on Robotics and Automation, v 2, pp 1095-1101, May 1992.

[6] Nielsen, L. F., Trostmann, M., Trostmann, E., and Conrad, F. "Robot Off-line Programming and Simulation As a True CIME-Subsystem." Proceedings of the 1992 IEEE International Conference on Robotics and Automation, v 2, pp 1089-1094, May 1992.

[7] Takahashi, T. and Sakai, T. "Teaching Robot's Movement in Virtual Reality." IEEE/ RSJ International Workshop on Intelligent Robots and Systems IROS '91, IEEE Cat. No. 91TH0375-6, v 2, pp 1583-1588, November 1991.

[8] Takahashi, T. and Ogata, H. "Robotic Assembly Operation based on Task-Level Teaching in Virtual Reality." Proceedings of the 1992 IEEE International Conference on Robotics and Automation, pp 1083-1088, May 1992.

[9] Brunner, B., Heindl, J., Hirzinger, G., and Landzettel, K. "Telerobotics Systems using Virtual Environment Display with Visual and Force Display Functions." Workshop S4, Force Display in Virtual Environments and its Application to Robotic Teleoperation, IEEE International Conference on Robotics and Automation, pp 98-111, May 1993.

[10] Backes, P. G., Beahan, J., and Bon, Bruce. "Interactive Command Building and Sequencing for Supervised Autonomy." Proceedings of the IEEE International Conference on Robotics and Automation, v 2, pp 795-801, May 1993.

[11] Mitsubishi Electric Corporation, *Industrial Micro-robot System Model RV-M1 Instruction Manual*, Tokyo, Japan, 1990.

[12] Troy, J. J. "An interactive graphical approach to off-line programming." M.S. Thesis, Iowa State University, Ames, IA, 1992.

[13] Craig, J. J. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, Reading, MA, 1986.

[14] Sense8 Corporation, *WorldToolKit Version 2.0 Reference Manual.* Sausalito, CA, 1993.

[15] Silicon Graphics Inc. *Graphics Library Programming Guide.* Mountain View, CA, 1991.

APPENDIX

Color images of VRS

- Path subsystem
- Task subsystem
- Movie subsystem
- Workcell subsystem

Sample files

- points.pts file
- workcell.obj file
- RV-M1 device control file rvm1.bas
- lights.lts file

Figure A.1 Path subsystem of *VRS*.

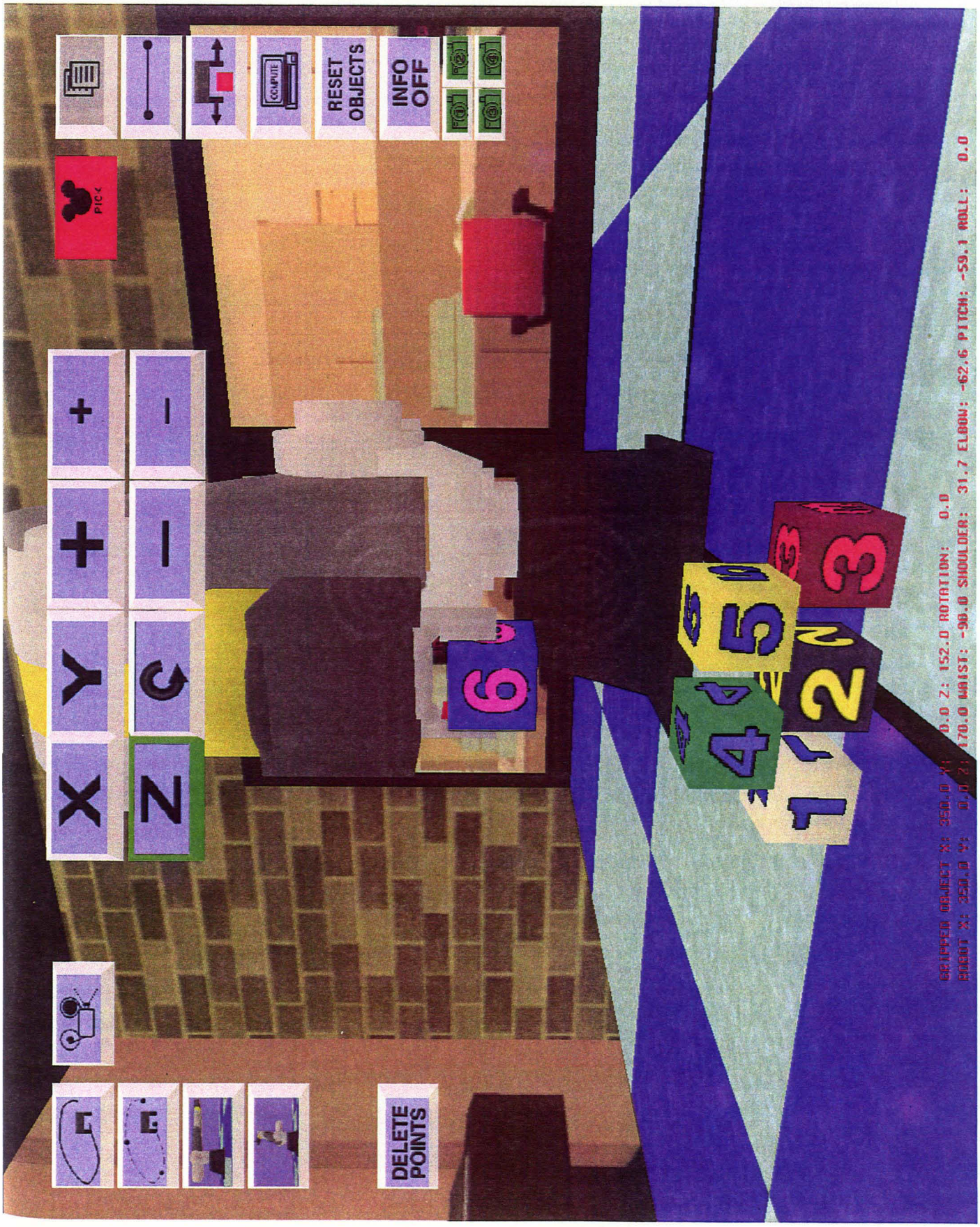
PICK
COMPUTE
RESET OBJECTS
INFO OFF
X
Y
Z
+
-
DELETE POINTS
GRIPPER OBJECT X: 350.0 Y: 0.0 Z: 152.0 ROTATION: 0.0
ROBOT X: 350.0 Y: 0.0 Z: 170.0 WAIST: -90.0 SHOULDER: 31.7 ELBOW: -62.6 PITCH: -59.1 ROLL: 0.0

Figure A.2 Task subsystem of *VRS*.

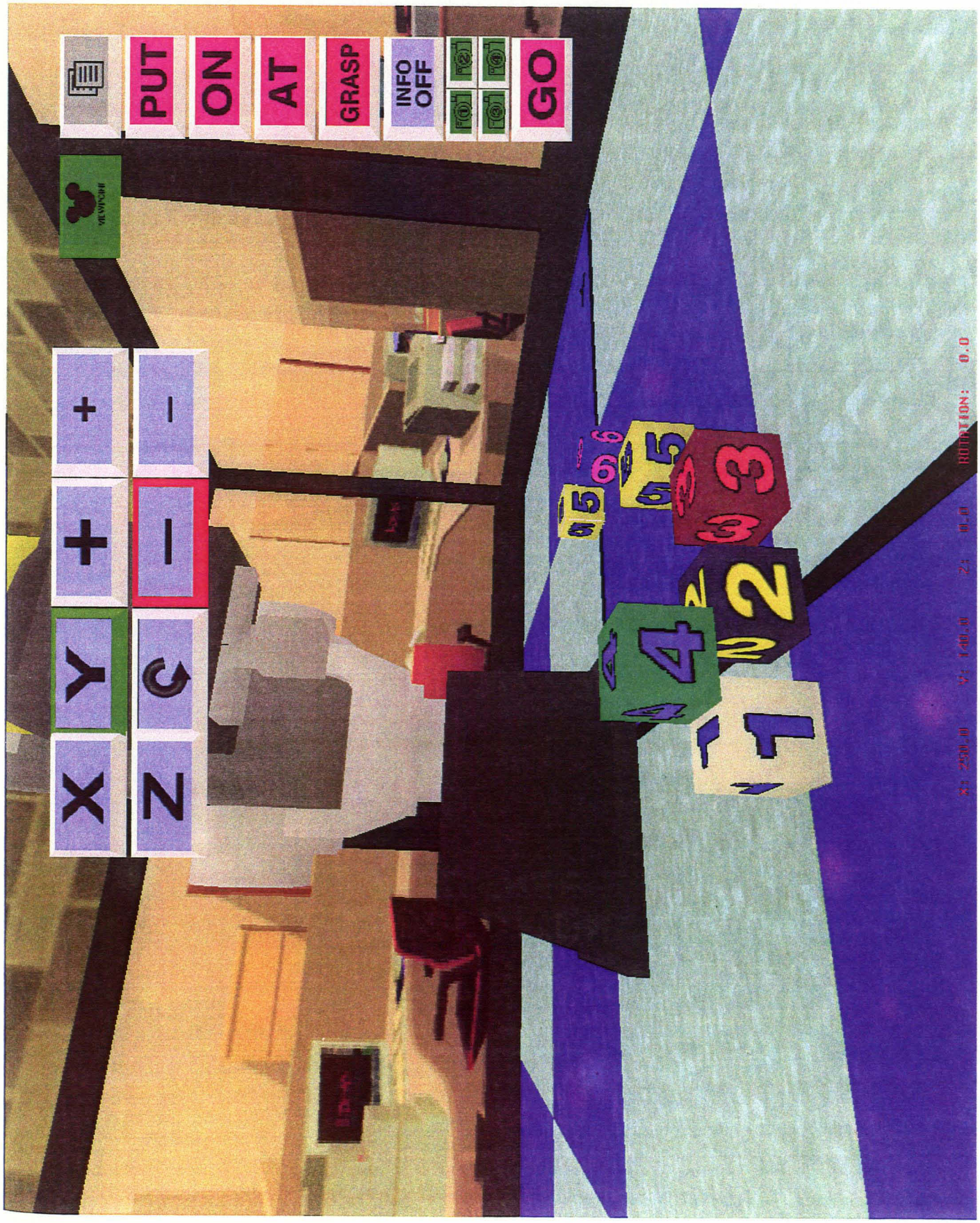
PUT
ON
AT
GRASP
INFO OFF
GO
X
Y
Z
C
+
-
+
-
+
-
VIEWPOINT
X: 250.0
Y: 140.0
Z: 0.0
ROTATION: 0.0

Figure A.3 Movie subsystem of *VRS*.

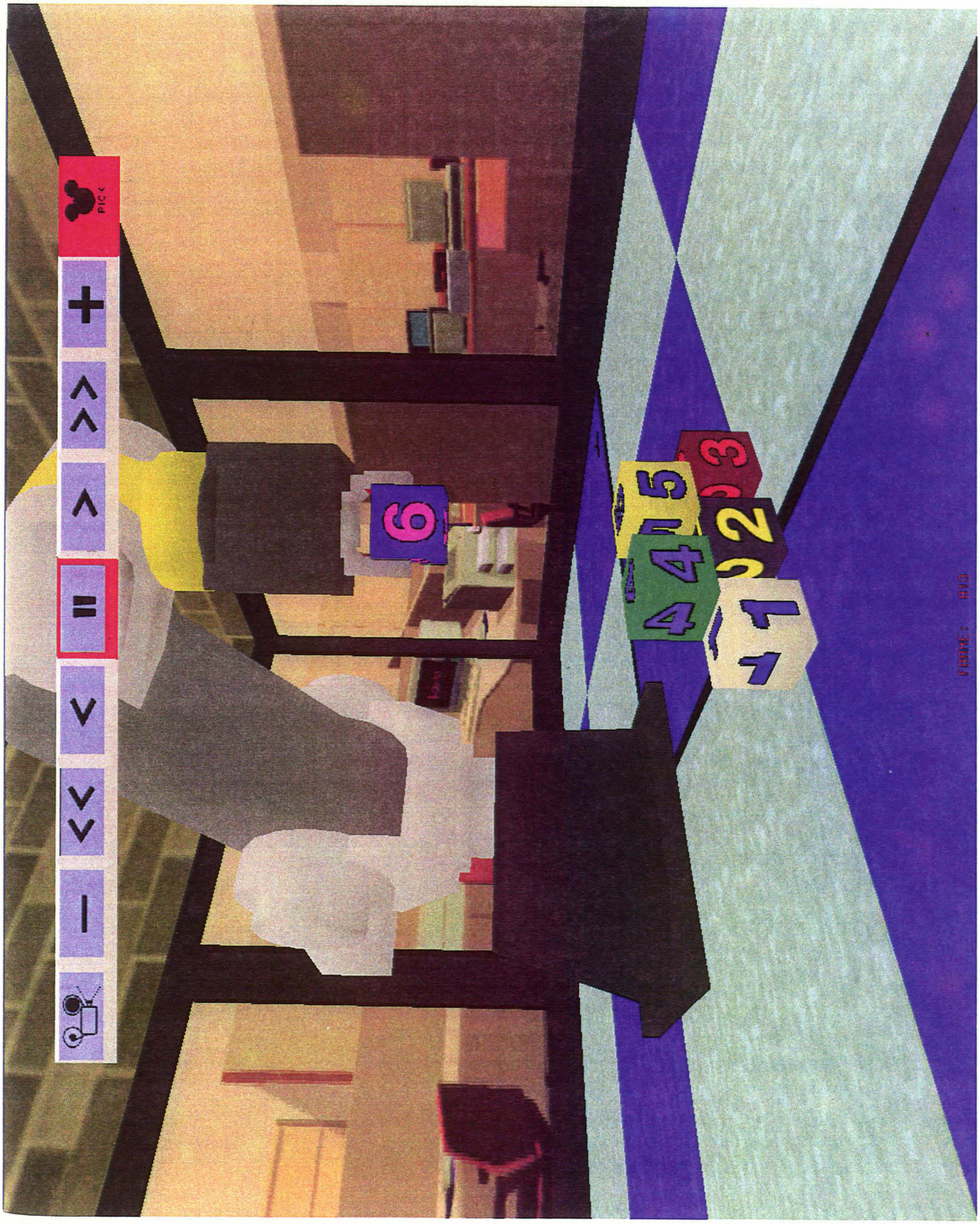
PICK
+
<
<<
=
>
>>
-
6
4
4
5
3
2
2
1
1

Figure A.4 Workcell subsystem of *VRS*.


PICK
OBJECTS
RED
GREEN
BLUE
INFO OFF
X
Y
Z
C
+
-
X: 100.0 Y: 250.0 Z: 0.0 ROTATION: 27.0

Sample Points File- points.pts

This is a sample points.pts file which saves the robot orientation at key locations along the path of the robot as the robot performs a task. A file of this type may be read into *VRS* and the path for the robot task may be computed. The robot performing the task may then be simulated.

PATH POINTS FILE points.pts

POINT	VEL	ACC	WAIST	SHOULDER	ELBOW	PITCH	ROLL	GRIPPER
1	9.00	1.00	0.000	0.000	0.000	1.571	0.000	0.000
2	9.00	1.00	-1.292	0.374	-0.982	0.608	0.000	0.000
3	9.00	1.00	-1.292	0.374	-0.982	0.608	0.000	30.000
4	9.00	1.00	-1.292	0.374	-0.982	0.608	1.292	30.000
5	9.00	1.00	-1.292	0.074	-0.846	0.772	1.292	30.000
6	9.00	1.00	-1.292	0.074	-0.846	0.772	1.292	19.900
7	9.00	1.00	-1.292	0.553	-0.885	0.333	1.292	19.900
8	9.00	1.00	-1.292	0.553	-0.885	0.333	1.292	19.900
9	9.00	1.00	-1.849	0.553	-0.885	0.333	1.292	19.900
10	9.00	1.00	-1.849	0.553	-0.885	0.333	1.849	19.900
11	9.00	1.00	-1.849	0.214	-0.940	0.725	1.849	19.900
12	9.00	1.00	-1.849	0.214	-0.940	0.725	1.849	30.000
13	9.00	1.00	-1.849	0.553	-0.885	0.333	1.849	30.000
14	9.00	1.00	-1.849	0.553	-0.885	0.333	1.849	0.000

Sample Workcell File- workcell.obj

This file saves the position, orientation, and color of the workcell objects used in *VRS*.

```
WORKCELL OBJECT FILE  workcell.obj

POSITION
 #      NAME       X          Y          Z
-------------------------------------------
 1      block1   350.000  -100.000    20.000
 2      block2   350.000     0.000    20.000
 3      block3   350.000   100.000    20.000
 4      block4  -350.000   100.000    20.000
 5      block5   300.000   120.000    20.000
 6      block6   150.000   200.000    20.000

ORIENTATION
 #      NAME       QX         QY         QZ         QW
-----------------------------------------------------
 1      block1   0.00000    0.00000    0.00000    1.00000
 2      block2   0.00000    0.00000    0.00000    1.00000
 3      block3   0.00000    0.00000    0.00000    1.00000
 4      block4   0.00000    0.00000    0.00000    1.00000
 5      block5   0.00000    0.00000    0.00000    1.00000
 6      block6   0.00000    0.00000    0.00000    1.00000

COLOR
 #      NAME       RED    GREEN    BLUE
-------------------------------------
 1      block1      12       12      10
 2      block2       0        0       3
 3      block3       4        0       0
 4      block4       2        9       5
 5      block5      12       12       3
 6      block6      13        9      10
```

Sample RV-M1 Device Control File- rvm1.dcf

This file saves a device control file written in BASIC which is used with the actual RV-M1 robot to perform the task that was programmed in *VRS*.

```
OPEN "COM2:9600,E,7,2,CS5000,DS5000" FOR RANDOM AS #2
PRINT #2, "PD 1,    0.0,  589.0,  300.0,    0.0,    0.0"
PRINT #2, "PD 2,  350.0,  100.0,  121.0,  -90.0,    0.0"
PRINT #2, "PD 3,  350.0,  100.0,  121.0,  -90.0,    0.0"
PRINT #2, "PD 4,  350.0,  100.0,  121.0,  -90.0,   74.1"
PRINT #2, "PD 5,  350.0,  100.0,   28.0,  -90.0,   74.1"
PRINT #2, "PD 6,  350.0,  100.0,   28.0,  -90.0,   74.1"
PRINT #2, "PD 7,  350.0,  100.0,  200.0,  -90.0,   74.1"
PRINT #2, "PD 8,  350.0,  100.0,  200.0,  -90.0,   74.1"
PRINT #2, "PD 9,  350.0, -100.0,  200.0,  -90.0,   74.1"
PRINT #2, "PD 10,  350.0, -100.0,  200.0,  -90.0,  105.9"
PRINT #2, "PD 11,  350.0, -100.0,   68.0,  -90.0,  105.9"
PRINT #2, "PD 12,  350.0, -100.0,   68.0,  -90.0,  105.9"
PRINT #2, "PD 13,  350.0, -100.0,  200.0,  -90.0,  105.9"
PRINT #2, "PD 14,  350.0, -100.0,  200.0,  -90.0,  105.9"
PRINT #2, "SP 9, H"
PRINT #2, "MO 1, C"
PRINT #2, "MO 2, C"
PRINT #2, "MO 3, O"
PRINT #2, "MO 4, O"
PRINT #2, "MO 5, O"
PRINT #2, "MO 6, C"
PRINT #2, "MO 7, C"
PRINT #2, "MO 8, C"
PRINT #2, "MO 9, C"
PRINT #2, "MO 10, C"
PRINT #2, "MO 11, C"
PRINT #2, "MO 12, O"
PRINT #2, "MO 13, O"
PRINT #2, "MO 14, C"
```

Sample Lights File- lights.lts

This file is the lighting file used in *VRS*. The location of the light is given by the first three numbers in the line. The next three numbers indicate the direction of the light. The last number gives the intensity of the light where 1.0 is the highest intensity and 0.0 is no intensity.

```
   0.0     0.0 200.0   0.0   0.0 -1.0 0.3
   0.0 -200.0   0.0   0.0   1.0   0.0 0.3
   0.0  200.0   0.0   0.0 -1.0   0.0 0.3
 200.0    0.0   0.0 -1.0   0.0   0.0 0.3
-200.0    0.0   0.0   1.0   0.0   0.0 0.3
```